L-Università ta' Malta
Faculty of Information & Communication Technology

CPS3232 Project: Applied Cryptography
Software Security Module (SSM)

Quentin Falzon, CS

February 2021

Contents

1 Introduction **4**

2 Implementation Overview **5**
- 2.1 Securing the Connection (TLS) 5
- 2.2 Express.js & the Crypto API 5
- 2.3 HTTP Request Handling 6
- 2.4 Project File Structure 6

3 SSM Administrator Functionality **7**
- 3.1 Mutual Authentication 7
- 3.2 The 2-Part Master Key 7
- 3.3 SSM Admin Methods & Variables *(/src/vault.js)* 8
  - 3.3.1 vaultExists(req, res) 9
  - 3.3.2 setupKeys(req, res) 9
  - 3.3.3 validateMK(req, res, i) 9
  - 3.3.4 loadVault(req, res) 10
  - 3.3.5 decryptAndMacKeyEncryptionKey(res) 10
  - 3.3.6 decryptAndMacApplicationKeysList(res) 10
  - 3.3.7 encryptAndMacKeyEncryptionKey() 10
  - 3.3.8 encryptAndMacApplicationKeysList() 10
  - 3.3.9 writeVault(res) 10
  - 3.3.10 createApplicationKey(req, res) 10
  - 3.3.11 isUniqueName(req) 11
  - 3.3.12 changeMasterKey(req, res) 11
  - 3.3.13 rotateKeyEncryptionKey(req, res) 11
  - 3.3.14 getApplicationKeyInfo(req, res) 11
  - 3.3.15 getApplicationKeysInfo(req, res) 11
  - 3.3.16 updateApplicationKey(req, res) 11
  - 3.3.17 isCompatibleConfig(req, res, ak) 11
- 3.4 Summary 12

4 SSM Application Functionality **12**
- 4.1 Mutual Authentication 12
- 4.2 Vault Decryption Without the Master Key 13
- 4.3 SSM Application Methods & Variables *(/src/vault.js)* 13
  - 4.3.1 decryptAndMacApplicationKeysList(res) 14
  - 4.3.2 rotateApplicationKey(ak) 14
  - 4.3.3 isExpired(ak) 14
  - 4.3.4 clientEncrypt(req, res, k, k1, k2) 14
  - 4.3.5 clientDecrypt(req, res, k, k1, k2) 14
- 4.4 Summary 14

5 Putting it All Together - UI **15**

6 Comparing Block Cipher Performance **22**
- 6.1 Tables of Results 22
  - 6.1.1 Encryption 22
  - 6.1.2 Decryption 22
- 6.2 Visualizing the Results 23

7 Enhancing the SSM Security **23**

8 Key Management Policy for the Master Key **24**

1 Introduction

The **hardware security module** (HSM) is a secure , tamper-resistant physical computing device that securely stores and manages cryptographic keys, and performs encryption and decryption using these keys for confidentiality. Using message authentication codes (MAC), it provides message authentication and integrity checks. The aim of this project is to write a software implementation of the HSM, called a **software security module** (SSM) which provides equivalent functionality.

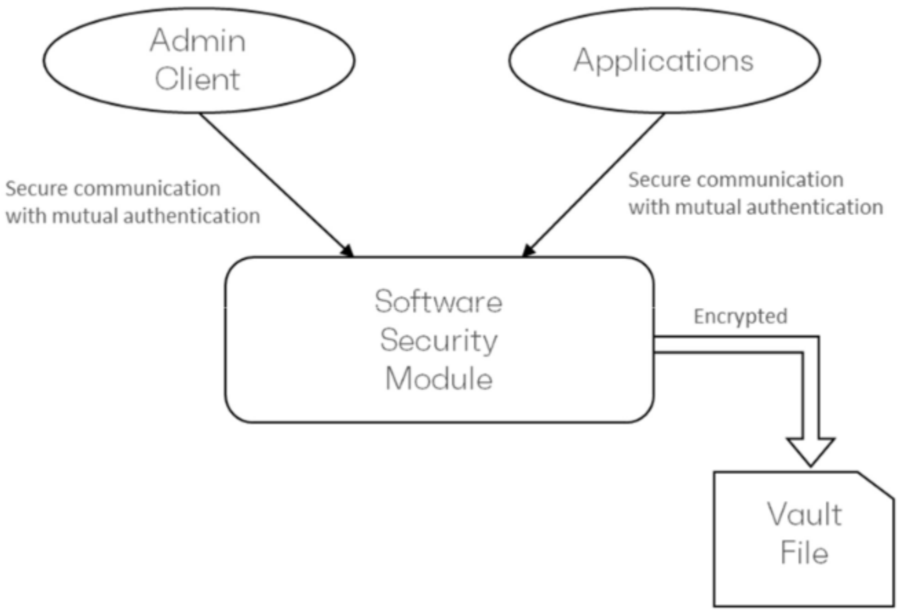

Admin Client
Applications
Secure communication with mutual authentication
Secure communication with mutual authentication
Software Security Module
Encrypted
Vault File

Figure 1: A high-level view of the SSM

The practical idea of the SSM is as follows. Suppose Alice has installed an open source messaging app *'Dart'* on her laptop, and wants to send a message to Bob, who is also using Dart. However, the app comes with the caveat of sending **unencrypted** messages. So Eve, a hypothetical eavesdropper would have no trouble capturing and making sense of conversations between Alice and Bob. The SSM solves this issue by posing as a point of encryption and decryption for Dart before it transmits over an insecure link e.g., the World Wide Web. Alice can secure her Dart app using the SSM by taking these steps:

- Create a Dart *application key* through the SSM administrator API.
- Configure Dart to use the SSM application API. Route all outbound messages through the **encrypt** endpoint and all inbound messages through the **decrypt** endpoint.

Now the Dart application key is created and exists encrypted and MACed inside the vault file. As long as Alice and Bob both have access to the **encrypt** and **decrypt** endpoints, they can use the SSM to encrypt and decrypt outbound and inbound Dart messages, respectively!

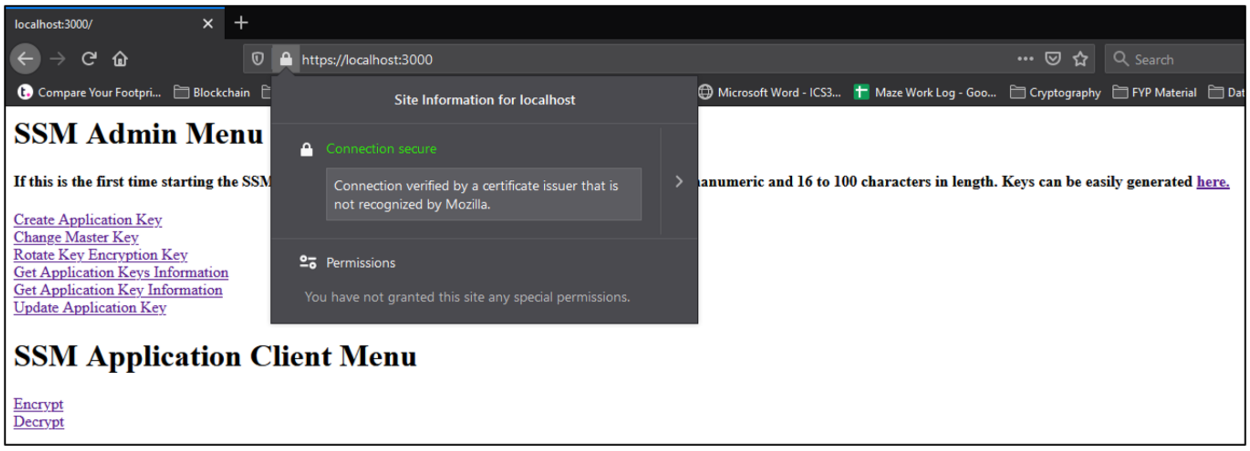
2 Implementation Overview

2.1 Securing the Connection (TLS)

I have outlined how the SSM can encrypt and decrypt application traffic, however there still remains the issue of securely routing plaintext Dart messages to and from the SSM. As mentioned, applications should be configured to route outbound messages through the encrypt endpoint, and inbound messages through the decrypt endpoint. Note that an outbound Dart message that has not yet been encrypted will be sent to the SSM in **plaintext**. Similarly, an inbound Dart message that has just been decrypted by the SSM will be sent to Dart in **plaintext**. Knowing this, Eve would still be able to intercept the plaintexts, making the SSM redundant. Therefore, a <u>secure connection</u> between the SSM and application client is paramount. The same issue exists for the admin API. For instance, when Alice creates the Dart application key, she must specify a **master key** to authenticate herself as an SSM administrator. As is, the master key would be sent to the SSM in plaintext, which is deeply undesirable. So a secure connection must be set for both the admin and pplication APIs.

2.2 Express.js & the Crypto API

I chose to tackle this project in NodeJS, using an Express.js[1] server and the official NodeJS Crypto API[2]. The API is based on *OpenSSL* and provides all encryption schemes, derivation functions, and cryptographic pseudo-randomness required for this project. Express.js allowed me to set up an HTTPS server which uses self-signed X.509 certificates to communicate over TLS 1.3. Note that these must be manually added as a trusted root authority within the browser. Now Eve is no longer able to read or modify messages sent through the application and admin APIs. A good starting point!



Site Information for localhost

Connection secure

Connection verified by a certificate issuer that is not recognized by Mozilla.

Permissions

You have not granted this site any special permissions.

SSM Admin Menu

If this is the first time starting the SSM **anumeric and 16 to 100 characters in length. Keys can be easily generated here.**

Create Application Key
Change Master Key
Rotate Key Encryption Key
Get Application Keys Information
Get Application Key Information
Update Application Key

SSM Application Client Menu

Encrypt
Decrypt

Figure 2: Connecting to the SSM securely over HTTPS

Express.js was also chosen due to its ease of use when handling HTTP requests and responses. By defining *req* and *res* objects, I can access anything sent by an admin or application. With JavaScript's asynchronous functions, it is easy to tell the server how to handle HTTP requests at different URLs.

[1]https://expressjs.com/
[2]https://nodejs.org/api/crypto.html

Another compelling reason to use JavaScript is **JavaScript Object Notation** (JSON), since the format of the vault file can be perfectly represented with key-value pairs. JSON can be serialized and deserialized extremely easily. Furthermore, a JSON object of any size and complexity can be encrypted effortlessly by calling *JSON.stringify(object)* to convert it to a string, then encrypting the string using a symmetric block cipher like AES.

Express.js supports *middleware* functions which can be called in sequence for every request, or only for specific requests. Examples of useful middleware functions which run on every request are:

- *helmet()*[3] - Ensures that HTTP headers are set appropriately and protects against any potential SSL-stripping attacks (Not an issue with TLS).
- *bodyParser()* - Parses data sent to the SSM and places it in *req* for easy access. Required for every admin and application operation.

2.3 HTTP Request Handling

The SSM is designed to handle GET and POST HTTP requests. Alice creates an application key through the admin API. Using the browser, she sends a GET request to *https://localhost:3000/admin/createapplicationkey*. The SSM handles the GET request by rendering back a *create_application_key* HTML form. Alice fills out the form, and submits it. The form submission event sends a POST request containing the form data (sensitive!), and the SSM handles this POST request by executing all the code associated with application key creation. An HTTP status code 200 is returned on successful execution. The request-handling logic is similar for all admin and application operations, since they all use forms.

2.4 Project File Structure

The project file structure is briefly outlined.

- **/src**
  - **key.js** - *Key* object constructor function. Used for all SSM keys.
  - **vault.js** - Contains **all** cryptographic and helper functions.
- **/views**
  - HTML files for rendering the form-based UI (admin & application APIs).
- **/www**
  - Certificate and key files necessary for mutual authentication .
- **index.js** - The SSM Express server. Renders appropriate page from **/views** on GET request. Calls appropriate function from **/src/vault.js** on POST request.
- **vault.json** - Encrypted vault file. Stores encrypted keys, MACs, salts and IVs.

[3]https://helmetjs.github.io/

3 SSM Administrator Functionality

3.1 Mutual Authentication

There is mutual authentication between the SSM administrator and the SSM. The admin authenticates themselves to the SSM using a **2-part master key**. A bad master key will result in unsuccessful vault decryption, thereby halting the admin operation. But how does the admin know they are not connected to a rogue SSM? The SSM must authenticate itself to the admin by presenting a certificate signed by a **Certificate Authority** (CA) which confirms their identity. Since the certificate is self-signed (i.e., I am the CA), Firefox does not recognize the CA, as shown in figure 2. Nonetheless, a fully secure HTTPS connection is established, and there is mutual authentication between the admin and SSM.

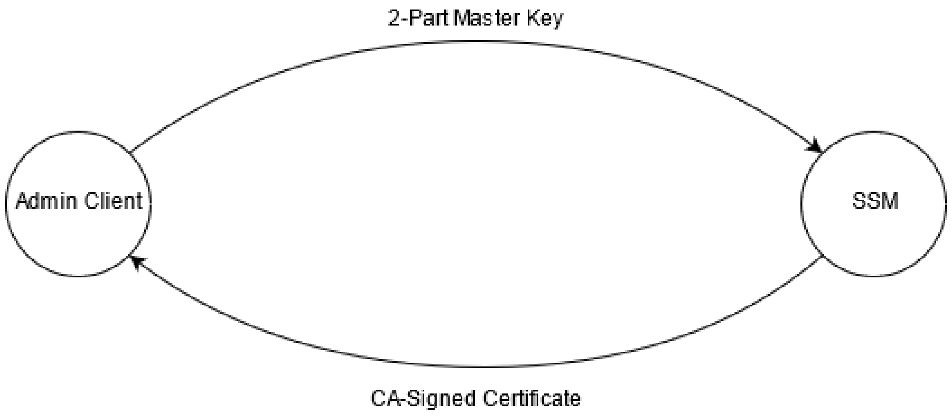


Figure 3: Admin-SSM mutual authentication

3.2 The 2-Part Master Key

It is worth noting that the master key is composed of two parts to prevent a single point of failure. The SSM should be administered by **two** admins, Alice and Bob, who each own a part of the master key, exclusively. No admin operation is possible unless both master key parts are provided. Therefore, if Alice's key is somehow compromised by Eve (through social engineering, bad key management policy etc.), this will not allow Eve to gain access to the SSM, as long as Bob keeps his part safe.

In this project I assume that Alice and Bob each inputs their master key part from the same machine, at the same time. In practice, this can be extended to support Alice and Bob authenticating from different locations, at different times.

3.3 SSM Admin Methods & Variables *(/src/vault.js)*

Before any application can connect to the SSM and encrypt/decrypt messages, an administrator must initialize the SSM and create an application key. Instructions on how to initialize the SSM can be found in the project README. The flowchart shows how admin operations are handled internally by the SSM.

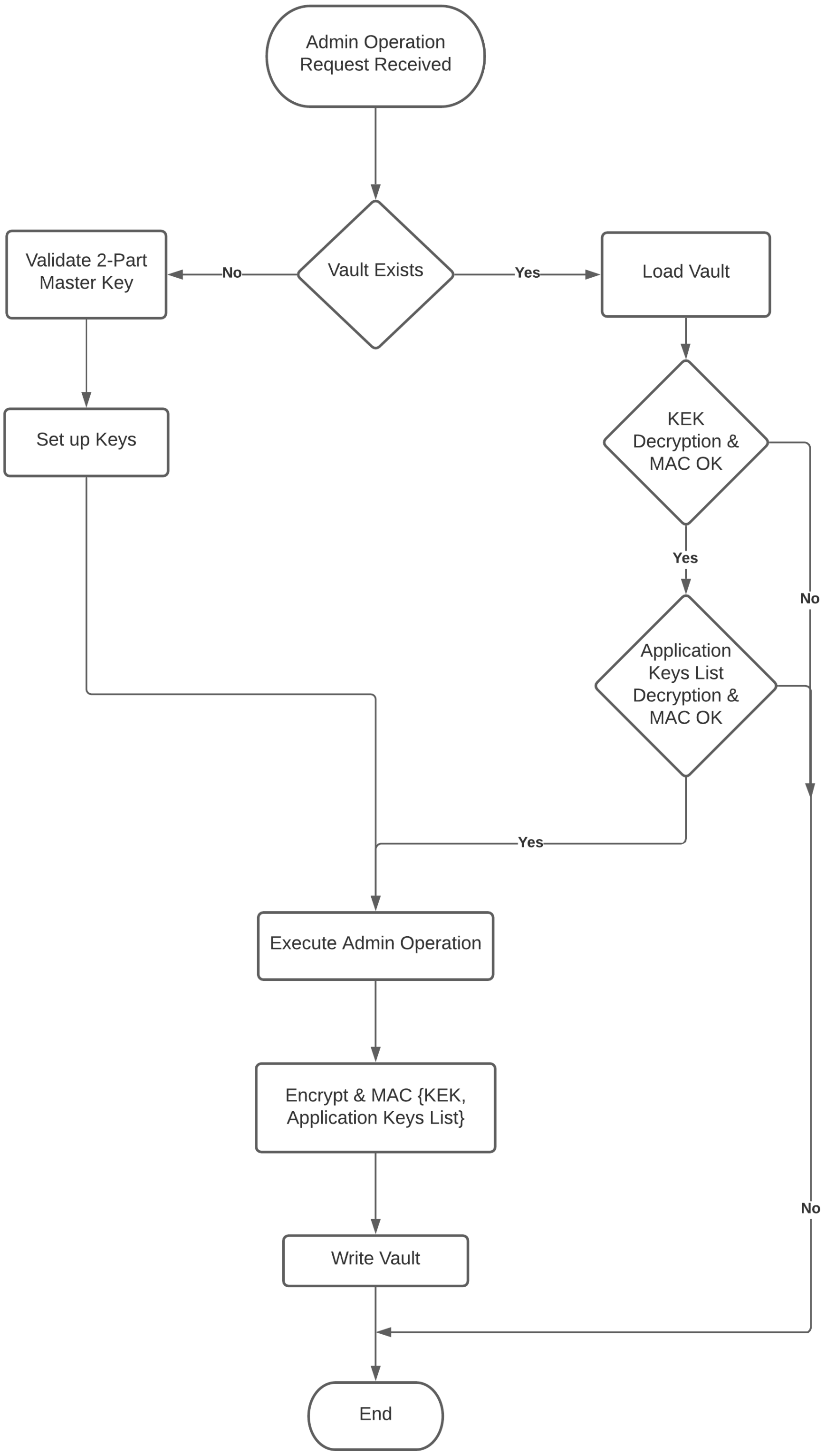

Admin Operation Request Received
Vault Exists
No
Yes
Validate 2-Part Master Key
Load Vault
Set up Keys
KEK Decryption & MAC OK
Yes
No
Application Keys List Decryption & MAC OK
Yes
Execute Admin Operation
Encrypt & MAC {KEK, Application Keys List}
No
Write Vault
End

Figure 4: Flowchart showing how every admin operation is handled internally

3.3.1 vaultExists(req, res)

As illustrated in figure 3, the SSM checks for an existing vault file every time an admin command is submitted. Calls **setupKeys()** if no vault file exists. If vault file exists, calls **loadVault()**. Returns the *KEK*, KEK_1, and KEK_2 key objects, so they can be referenced by the application API within **index.js**.

3.3.2 setupKeys(req, res)

Calls **validateMK()** to validate the first-time master key. Upon success, it assigns values to the pre-declared global variables:

- Concatenate 2-part master key to obtain the master key *MK*
- Generate **cryptographically secure** pseudo-random 16-byte salts $\{saltMK_1, saltMK_2\}$.
- $MK_1.value$ = pbkdf2(MK, $saltMK_1$) using SHA512 for 100,000 iterations.
- $MK_1.iv$ = random 16-byte IV.
- $MK_2.value$ = pbkdf2(MK, $saltMK_2$) using SHA512 for 100,000 iterations.
- $KEK.value$ = **cryptographically secure** pseudo-random 32 bytes.
- Generate **cryptographically secure** pseudo-random 16-byte salts $\{saltKEK_1, saltKEK_2\}$
- $KEK_1.value$ = pbkdf2(KEK, $saltKEK_1$) using SHA512 for 100,000 iterations.
- $KEK_1.iv$ = random 16-byte IV.
- $KEK_2.value$ = pbkdf2(KEK, $saltKEK_2$) using SHA512 for 100,000 iterations.

There are six remaining pre-declared global variables which are not yet set. These will get set accordingly as the admin operation runs.

- *KEKEncMK1* - A string which stores the AES-256-CBC encrypted KEK.
- *KEKMacMK2* - A string which stores the SHA256 HMACed KEK.
- *vault* - An empty object which will be populated when loading or persisting the vault.
- *applicationkeys* - An empty array which stores application keys as objects.
- *applicationkeysEncKEK1* - An empty string which stores the encrypted application keys list.
- *applicationkeysMacKEK2* - An empty string which stores the MAC of the unencrypted application keys list.

3.3.3 validateMK(req, res, i)

Validates the 2-part master key against the following constraints:

- Each part must be between 16 and 100 characters in length.
- Each part must contain lowercase letters.
- Each part must contain digits.
- There must not be any spaces.

3.3.4 loadVault(req, res)

Reads the vault file from storage as a long string and parses it into an object. Populates the variables similarly to **setupKeys()**, but salts and IVs are read from vault. Calls **decryptAndMacKeyEncryptionKey()**, derives $\{KEK_1.value, KEK_2.value\}$, and subsequently calls **decryptAndMacApplicationKeysList()**.

3.3.5 decryptAndMacKeyEncryptionKey(res)

Decrypts the *KEK* from vault, with the derived $MK_1.value$ as key and $MK_1.iv$ fetched from the vault as IV. Stores result in *KEK.value*. Computes the SHA256 HMAC of the decrypted KEK with MK_2 as key. Asserts that newly computed digest === digest read from vault. Otherwise, writes error and 401 HTTP status to *res*, halting the admin operation.

3.3.6 decryptAndMacApplicationKeysList(res)

Decrypts the application keys list from vault, with the derived $KEK_1.value$ as key and $KEK_1.iv$ fetched from the vault as IV. Parses result into an array and stores it in *applicationkeys*. Computes the SHA256 HMAC digest of the decrypted KEK with KEK_2 as key. Asserts that newly computed digest === digest read from vault. Otherwise, writes error and 401 HTTP status to *res*, halting the admin operation.

3.3.7 encryptAndMacKeyEncryptionKey()

Encrypts the *KEK*, using AES-256-CBC, with $MK_1.value$ as key and $MK_1.iv$ as IV. Stores result in *KEKEncMK1* string. Stores result in *KEKEncMK1.* Computes the SHA256 HMAC of *KEK.value* with MK_2 as key. Stores result in *KEKMacMK2.*

3.3.8 encryptAndMacApplicationKeysList()

Converts *applicationkeys* array into a string and encrypts the string with $KEK_1.value$ as key and $KEK_1.iv$ as IV. Stores result in *applicationkeysEncKEK1.* Computes the SHA256 HMAC of the unencrypted string with $KEK_2.value$ as key. Stores result in *applicationkeysMacKEK2.*

3.3.9 writeVault(res)

encryptAndMacKeyEncryptionKey() and **encryptAndMacApplicationKeysList()** are called first. Populates the *vault* object, converts it to string, and writes the string to disk.

3.3.10 createApplicationKey(req, res)

Calls **isUniqueName()** to ensure the provided name will not cause a conflict. Creates a new *Key* object. Generates an appropriately sized, **cryptographically secure** pseudo-random DES, 3DES or AES key and IV if using any mode other than ECB. Sets *name, algorithm, mode, value, iv, padding, keysize* and *autorotate* attributes. Sets *lifetime* to a *Date* object.

3.3.11 isUniqueName(req)

Returns true if there is no existing application key within the SSM with a name matching *req.body.name*. Otherwise returns false.

3.3.12 changeMasterKey(req, res)

Calls **validateMK()** to validate the new 2-part master key. Generates and sets new **cryptographically secure** pseudo-random 16-byte salts $\{saltMK_1,\ saltMK_2\}$. Concatenates new 2-part master key to obtain master key *MK*. Derives new $MK_1.value$ using pbkdf2(*MK*, $saltMK_1$) with SHA512 for 100,000 iterations. Generates **cryptographically secure** pseudo-random 16-byte IV for $MK_1.iv$. Derives and sets new $MK_2.value$ using pbkdf2(*MK*, $saltMK_2$) with SHA512 for 100,000 iterations.

3.3.13 rotateKeyEncryptionKey(req, res)

Generates and sets a new **cryptographically secure** pseudo-random 16-byte *KEK.value*. Generates and sets new **cryptographically secure** pseudo-random 16-byte salts $\{saltKEK_1,\ saltKEK_2\}$. Derives and sets new $KEK_1.value$ using pbkdf2(*KEK.value*, $saltKEK_1\}$. Generates **cryptographically secure** pseudo-random 16-byte IV for $KEK_1.iv$. Derives and sets new $KEK_2.value$ using pbkdf2(*KEK.value*, $saltKEK_2\}$.

3.3.14 getApplicationKeyInfo(req, res)

Searches *applicationkeys* array for a key where *Key.name* matches *req.body.name*. Returns the object as JSON through *res*. If no matching Key object is found, writes error and 404 HTTP status to *res*.

3.3.15 getApplicationKeysInfo(req, res)

Returns the entire *applicationkeys* array through *res*. If the array is empty, writes error and 404 HTTP status to *res*.

3.3.16 updateApplicationKey(req, res)

Searches *applicationkeys* array for a key where *Key.name* matches *req.body.name*. If found, calls **isCompatibleConfig()** and updates the application key parameters accordingly. If no matching Key object is found, writes error and 404 HTTP status to *res*.

3.3.17 isCompatibleConfig(req, res, ak)

Ensures that admin is not attempting to change to an invalid set of parameters. For example, the crypto API does not support DES and 3DES in GCM or CTR mode, so if the admin is trying to set a DES or 3DES application key to use either modes, an error and 401 HTTP status will be written to *res*. If configuration is OK, return true.

3.4 Summary

- The master key is kept in memory for the absolute shortest possible amount of time (< 1 second). It is never stored, and is never assigned to any other variable except *req*. The *req* variable is **automatically flushed by Express** as soon as a response is received. Responses typically take no longer than a second!
- The KEK and *applicationkeys* array are always encrypted using AES-256-CBC.
- The KEK and *applicationkeys* array are always HMACed using SHA256.
- For application keys, admins Alice and Bob can use the algorithms:
  - DES
  - 3DES (keying option 2)
  - 3DES (keying option 3)
  - AES
- All application keys can be configured to use {ECB, CFB-1, CFB-8, CFB, OFB, CBC}.
- AES keys can also use {CTR, GCM}.

4 SSM Application Functionality

4.1 Mutual Authentication

How does an application know it is routing messages to and from the real SSM? Conversely, how does the SSM know it is providing encryption and decryption services to the real application? It is abundantly clear that here too, there must be mutual authentication between an application (e.g., Dart) and the SSM. Otherwise, Eve could set up a rogue SSM and tamper with the data sent by Dart in a man-in-the-middle fashion. Worse still, Eve could design a rogue application and use the encrypt/decrypt API maliciously.

Mutual authentication is achieved using certificates both ways. When trying to use the API, Dart can view the SSM's CA-signed certificate. Since this certificate was added as a trusted root authority in the browser, the SSM is authenticated. At this point, Dart is prompted to provide a certificate, signed by the CA. If Dart is unable to produce such a certificate, then it is not granted access to the API. Otherwise, mutual authentication has succeeded.

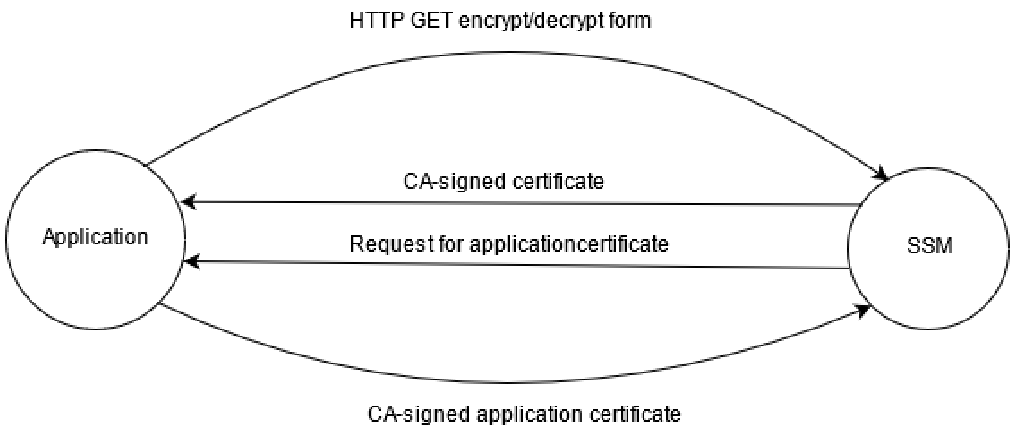

HTTP GET encrypt/decrypt form
CA-signed certificate
Application
Request for applicationcertificate
SSM
CA-signed application certificate

Figure 5: Application-SSM mutual authentication

4.2 Vault Decryption Without the Master Key

To perform encryption and decryption, an application needs to access its Key object from the *applicationkeys* global SSM array. A naive way to ensure the array is always available is by keeping a copy of of it in **index.js**. This way, as long as the Express server is online, the array is accessible. However, this exposes the keys unnecessarily and indefinitely, which is deeply undesirable.

Instead, the SSM and vault are designed in such a way that the *applicationkeys* array can be read and encrypted/decrypted/HMACed **without** the master key. This is achieved by keeping a copy of the KEK in **index.js**. Although it is a trade-off, I prefer exposing the single KEK as opposed to all application keys! Now, the keys' exposure is limited to whenever the application API is in use. All other times, they are stored encrypted in the vault, instead of in a local unencrypted variable.

4.3 SSM Application Methods & Variables *(/src/vault.js)*

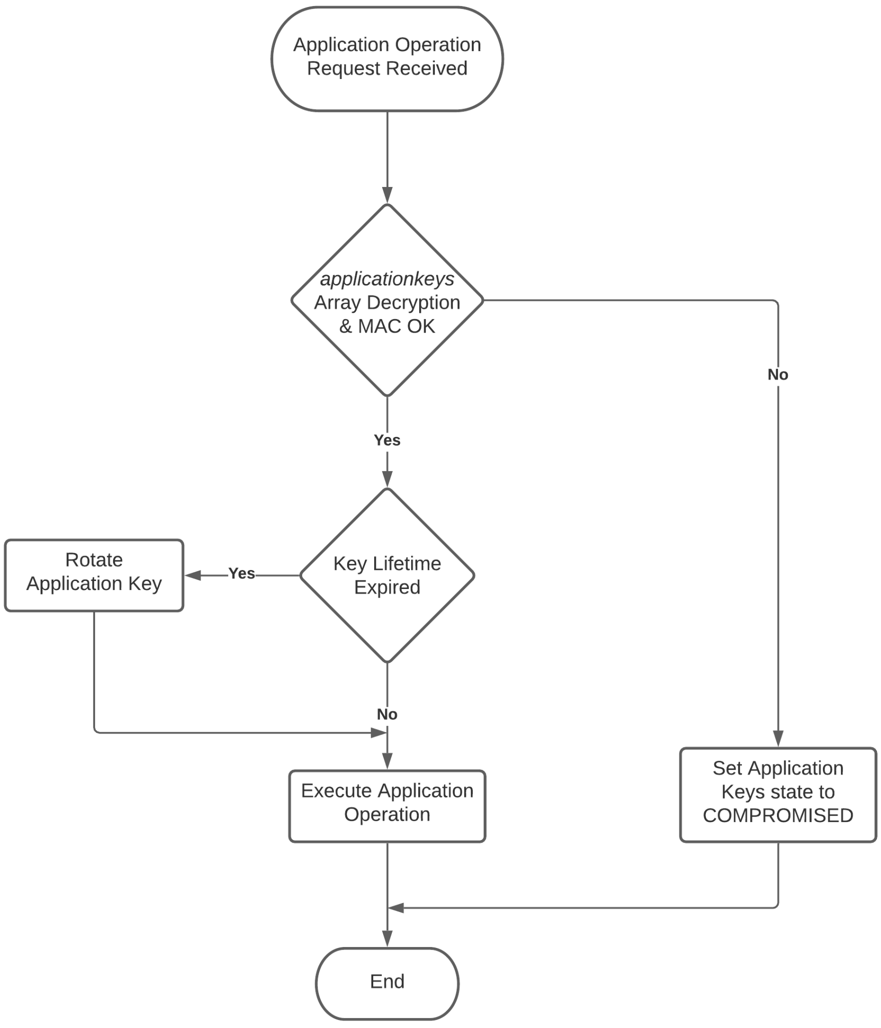

Application Operation Request Received
applicationkeys Array Decryption & MAC OK
No
Yes
Rotate Application Key
Yes
Key Lifetime Expired
No
Execute Application Operation
Set Application Keys state to COMPROMISED
End

Figure 6: Flowchart showing how every application operation is handled internally. Note that after successful execution, the decrypted *applicationkeys* array is not reencrypted, but simply garbage-collected from memory once the function returns.

The application API is more straightforward the admin API, as illustrated by figure 6. Once mutual authentication is established, the SSM fetches the encrypted *applicationkeys* array from vault and decrypts it. If the encrypted array has been tampered with, the decryption and HMAC will fail, and a nasty error message will be thrown.

4.3.1 decryptAndMacApplicationKeysList(res)

Refer to sub-subsection 3.3.6

4.3.2 rotateApplicationKey(ak)

Generates and sets an appropriately sized **cryptogrpahically secure** pseudo-random DES, 3DES or AES key and IV if using any other mode than ECB. Sets *name, algorithm, mode, value, iv, padding, keysize* and *autorotate* attributes. Sets *lifetime* to a *Date* object.

4.3.3 isExpired(ak)

Accepts a *Key* object as parameter, and compares the *Date* object at *ak.lifetime* to *Date*.now(). If *lifetime* is in the past, the application key is expired, returns true. Otherwise, returns false.

4.3.4 clientEncrypt(req, res, k, k1, k2)

Accepts the KEK, KEK_1, and KEK_2 as parameters in addition to *req* and *res.* Calls **decryptAndMacApplicationKeysList()** to get *applicationkeys* array of *Key* objects. Searches through array to find application key with name matching *req.body.name.* If no match is found, writes error and 401 HTTP status to *res.* Otherwise, encrypts application data (*req.body.data)* and writes it to *res.*

4.3.5 clientDecrypt(req, res, k, k1, k2)

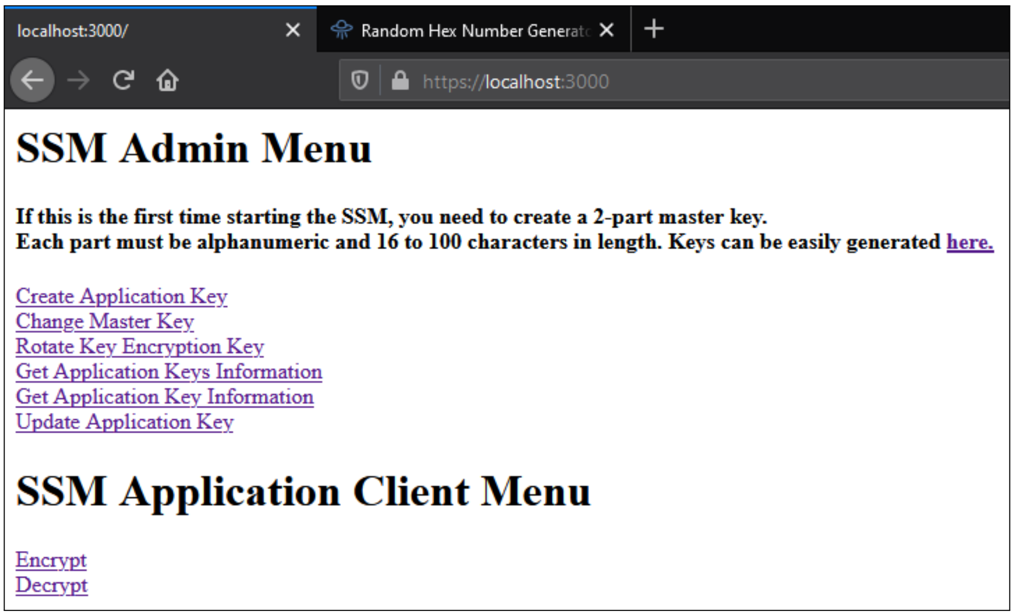
Same logic as **clientEncrypt()**, but uses the application key to perform decryption instead of encryption.

4.4 Summary

- An admin operation must be successfully executed before an application can use the application API. This is because copies of the KEK, KEK_1 and KEK_2 are only made available in **index.js** once an admin operation executes.
- If the encrypted *applicationkeys* array is tampered with, the decryption and HMAC will throw a nasty-looking error through *res.* Such errors are meant to be thrown, but can be handled more smoothly. These finishing touches were skipped in the interest of time.

5 Putting it All Together - UI

A simple front-end user interface was developed to interact with the SSM through the admin and application APIs.



localhost:3000/ | Random Hex Number Generat

https://localhost:3000

SSM Admin Menu

If this is the first time starting the SSM, you need to create a 2-part master key.
Each part must be alphanumeric and 16 to 100 characters in length. Keys can be easily generated here.

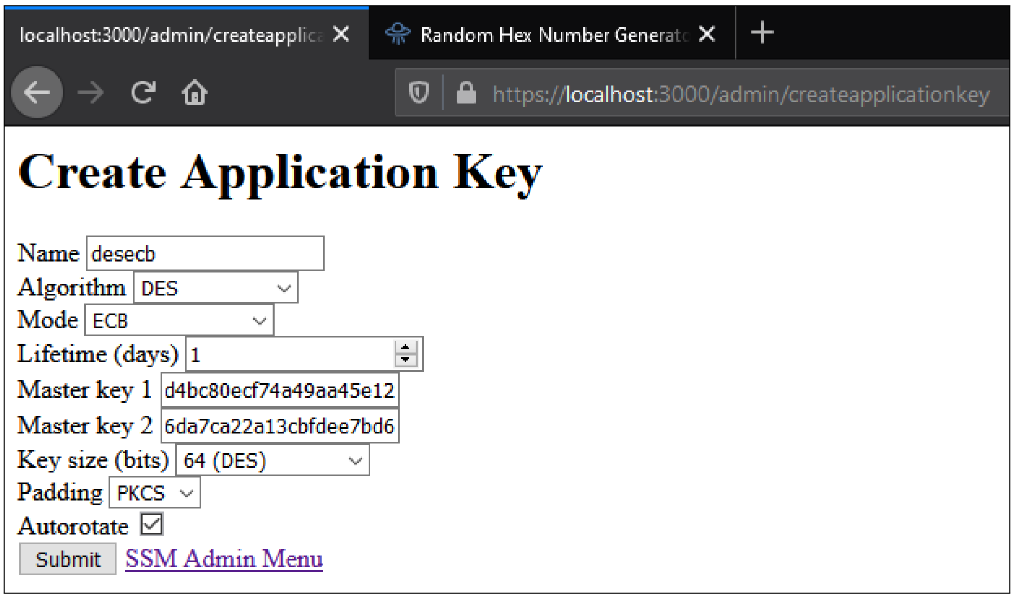
Create Application Key
Change Master Key
Rotate Key Encryption Key
Get Application Keys Information
Get Application Key Information
Update Application Key

SSM Application Client Menu

Encrypt
Decrypt

Figure 7: SSM main menu

i begin by creating an application key. I generate a random hexadecimal 2-part master key, since this is the first time the SSM is starting.



localhost:3000/admin/createapplic | Random Hex Number Generat

https://localhost:3000/admin/createapplicationkey

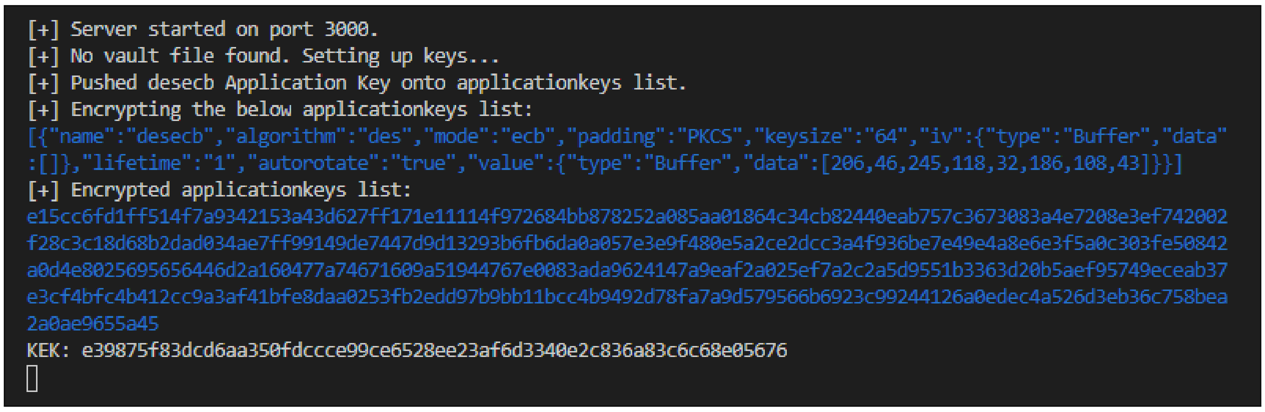
Create Application Key

Name	desecb
Algorithm	DES
Mode	ECB
Lifetime (days)	1
Master key 1	d4bc80ecf74a49aa45e12
Master key 2	6da7ca22a13cbfdee7bd6
Key size (bits)	64 (DES)
Padding	PKCS
Autorotate	☑

Submit | SSM Admin Menu

Figure 8: Creating a DES-ECB key

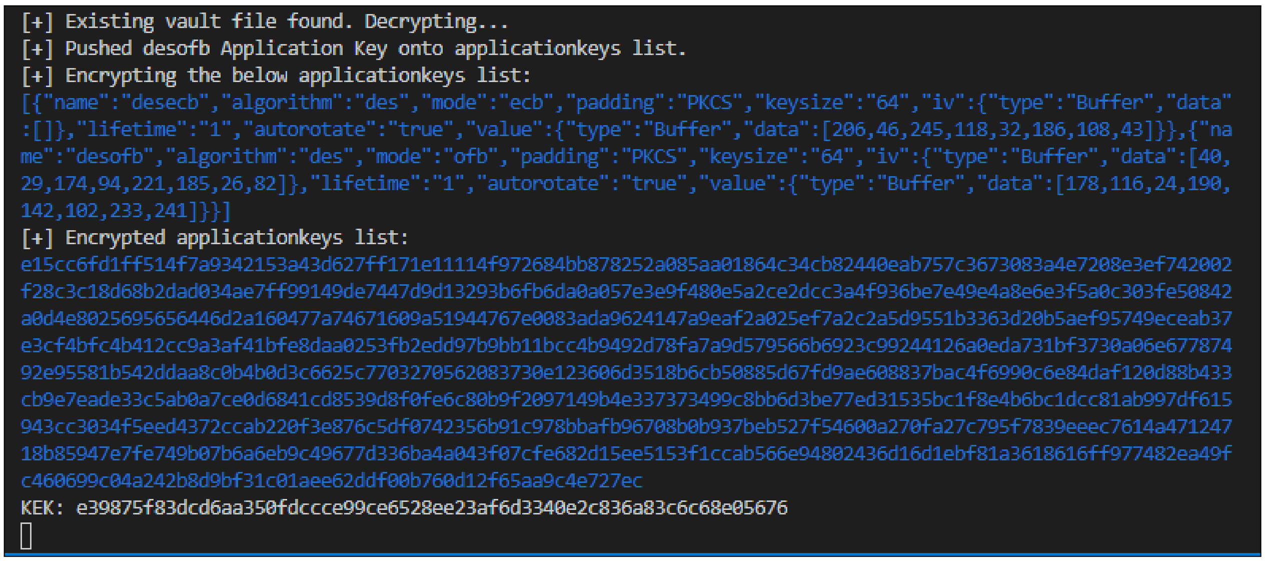
The SSM outputs useful information as it executes the admin command:



```
[+] Server started on port 3000.
[+] No vault file found. Setting up keys...
[+] Pushed desecb Application Key onto applicationkeys list.
[+] Encrypting the below applicationkeys list:
[{"name":"desecb","algorithm":"des","mode":"ecb","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data"
:[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[206,46,245,118,32,186,108,43]}}]
[+] Encrypted applicationkeys list:
e15cc6fd1ff514f7a9342153a43d627ff171e11114f972684bb878252a085aa01864c34cb82440eab757c3673083a4e7208e3ef742002
f28c3c18d68b2dad034ae7ff99149de7447d9d13293b6fb6da0a057e3e9f480e5a2ce2dcc3a4f936be7e49e4a8e6e3f5a0c303fe50842
a0d4e8025695656446d2a160477a74671609a51944767e0083ada9624147a9eaf2a025ef7a2c2a5d9551b3363d20b5aef95749eceab37
e3cf4bfc4b412cc9a3af41bfe8daa0253fb2edd97b9bb11bcc4b9492d78fa7a9d579566b6923c99244126a0edec4a526d3eb36c758bea
2a0ae9655a45
KEK: e39875f83dcd6aa350fdccce99ce6528ee23af6d3340e2c836a83c6c68e05676
```

Figure 9: SSM verbose output

Upon adding a second application key, the SSM finds an existing vault file, decrypts its contents, pushes the new key object onto the list, and re-encrypts everything as shown.

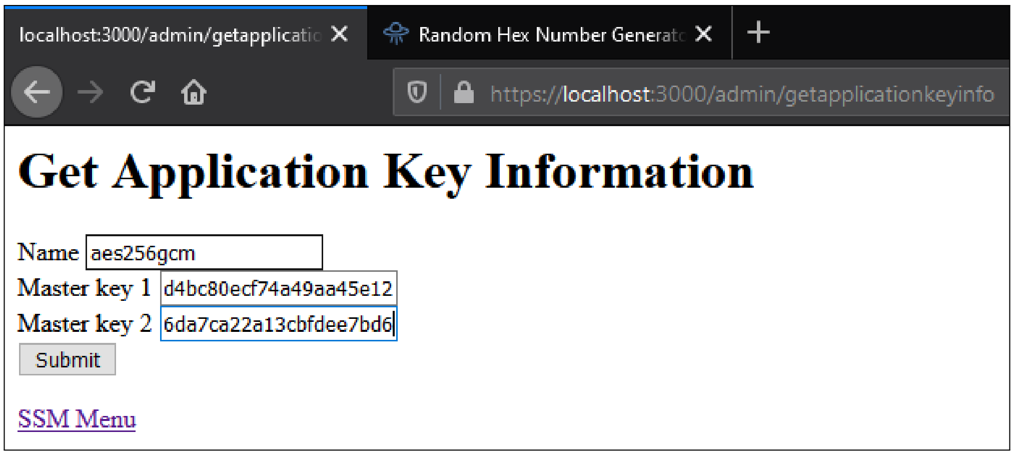


```
[+] Existing vault file found. Decrypting...
[+] Pushed desofb Application Key onto applicationkeys list.
[+] Encrypting the below applicationkeys list:
[{"name":"desecb","algorithm":"des","mode":"ecb","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data"
:[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[206,46,245,118,32,186,108,43]}},{"na
me":"desofb","algorithm":"des","mode":"ofb","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data":[40,
29,174,94,221,185,26,82]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[178,116,24,190,
142,102,233,241]}}]
[+] Encrypted applicationkeys list:
e15cc6fd1ff514f7a9342153a43d627ff171e11114f972684bb878252a085aa01864c34cb82440eab757c3673083a4e7208e3ef742002
f28c3c18d68b2dad034ae7ff99149de7447d9d13293b6fb6da0a057e3e9f480e5a2ce2dcc3a4f936be7e49e4a8e6e3f5a0c303fe50842
a0d4e8025695656446d2a160477a74671609a51944767e0083ada9624147a9eaf2a025ef7a2c2a5d9551b3363d20b5aef95749eceab37
e3cf4bfc4b412cc9a3af41bfe8daa0253fb2edd97b9bb11bcc4b9492d78fa7a9d579566b6923c99244126a0eda731bf3730a06e677874
92e95581b542ddaa8c0b4b0d3c6625c7703270562083730e123606d3518b6cb50885d67fd9ae608837bac4f6990c6e84daf120d88b433
cb9e7eade33c5ab0a7ce0d6841cd8539d8f0fe6c80b9f2097149b4e337373499c8bb6d3be77ed31535bc1f8e4b6bc1dcc81ab997df615
943cc3034f5eed4372ccab220f3e876c5df0742356b91c978bbafb96708b0b937beb527f54600a270fa27c795f7839eeec7614a471247
18b85947e7fe749b07b6a6eb9c49677d336ba4a043f07cfe682d15ee5153f1ccab566e94802436d16d1ebf81a3618616ff977482ea49f
c460699c04a242b8d9bf31c01aee62ddf00b760d12f65aa9c4e727ec
KEK: e39875f83dcd6aa350fdccce99ce6528ee23af6d3340e2c836a83c6c68e05676
```

Figure 10: Adding a second application key (DES-OFB)

A key was added for each key type and mode of operation.

To get information about a particular application key, the key name is specified, along with the 2-part master key.

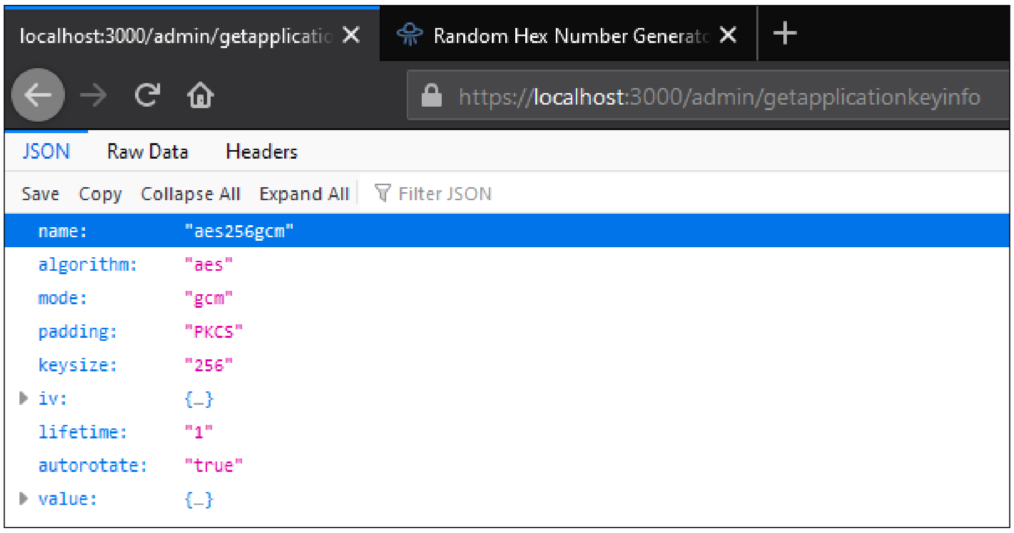

localhost:3000/admin/getapplicatio
Random Hex Number Generato
https://localhost:3000/admin/getapplicationkeyinfo

Get Application Key Information

Name aes256gcm
Master key 1 d4bc80ecf74a49aa45e12
Master key 2 6da7ca22a13cbfdee7bd6

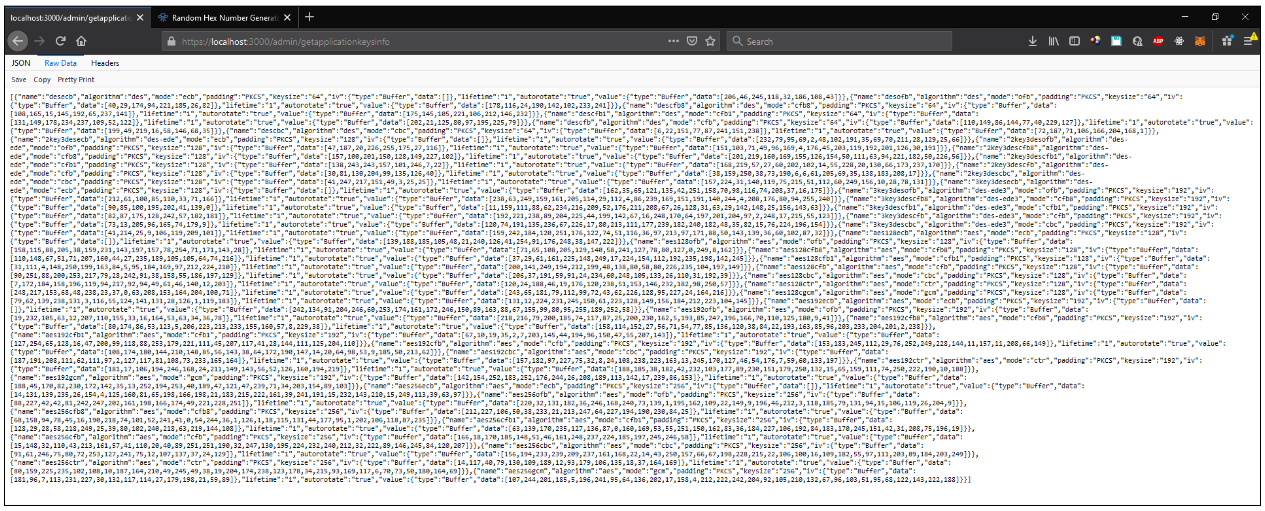
Figure 11: Retrieving information for the AES-256-GCM application key

The SSM responds by sending the JSON key object back. It is worth noting that the key value and IV are **not** in plaintext.


localhost:3000/admin/getapplicatio
Random Hex Number Generato
https://localhost:3000/admin/getapplicationkeyinfo
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
name:        "aes256gcm"
algorithm:   "aes"
mode:        "gcm"
padding:     "PKCS"
keysize:     "256"
iv:          {…}
lifetime:    "1"
autorotate:  "true"
value:       {…}
```

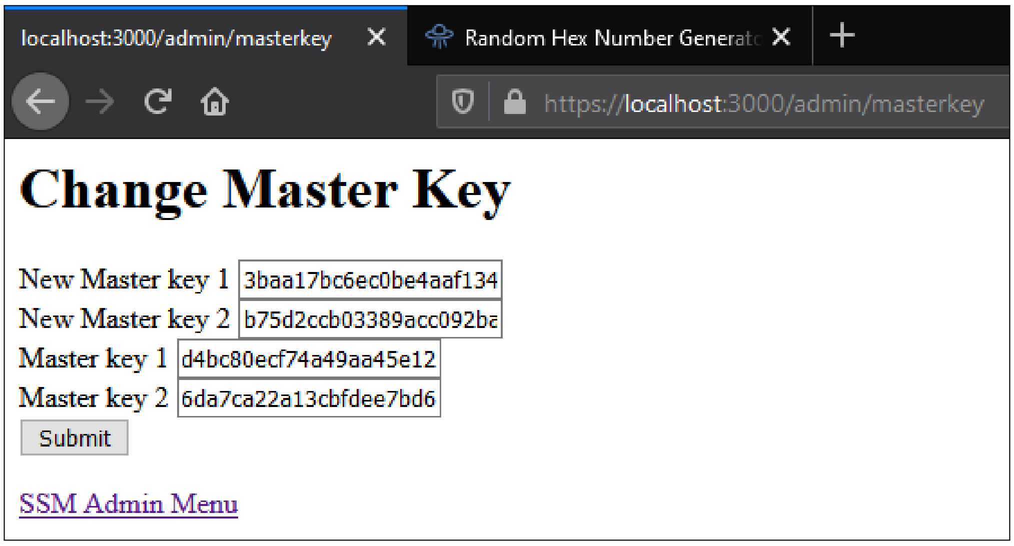
Getting information for all application keys works the exact same way - a JSON object is returned. However, this time the JSON object is the entire *applicationkeys* data structure. Since I have added a key of each algorithm and mode of operation, the JSON object is quite large. Displaying it as raw data:


localhost:3000/admin/getapplicatio
Random Hex Number Generato
https://localhost:3000/admin/getapplicationkeysinfo
Search
JSON
Raw Data
Headers
Save
Copy
Pretty Print

```
[{"name":"desecb","algorithm":"des","mode":"ecb","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data":[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[206,46,245,118,32,186,108,43]}},{"name":"desofb","algorithm":"des","mode":"ofb","padding":"PKCS","keysize":"64","iv":
{"type":"Buffer","data":[40,29,174,94,221,185,26,82]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[178,116,24,190,142,102,233,241]}},{"name":"descfb8","algorithm":"des","mode":"cfb8","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data":
[108,165,15,145,192,65,237,141]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[175,145,105,221,106,212,146,232]}},{"name":"descfb1","algorithm":"des","mode":"cfb1","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data":
[131,149,178,234,237,109,52,122]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[202,21,125,80,97,195,225,79]}},{"name":"descfb","algorithm":"des","mode":"cfb","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data":[110,149,86,144,77,40,229,127]},"lifetime":"1","autorotate":"true","value":
{"type":"Buffer","data":[199,49,219,16,58,146,68,35]}},{"name":"descbc","algorithm":"des","mode":"cbc","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data":[6,22,151,77,87,241,151,238]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[72,187,71,106,166,204,168,1]}},
{"name":"2key3desecb","algorithm":"des-ede","mode":"ecb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[232,79,95,69,2,48,102,191,35,69,70,211,28,129,25,66]}},{"name":"2key3desofb","algorithm":"des-
ede","mode":"ofb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[47,187,20,226,255,175,27,116]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[151,103,71,49,96,169,4,176,45,203,119,192,201,126,30,191]}},{"name":"2key3descfb8","algorithm":"des-
ede","mode":"cfb8","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[157,100,201,150,128,149,227,102]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[201,219,160,169,155,126,154,50,111,63,94,221,182,50,226,56]}},{"name":"2key3descfb1","algorithm":"des-
ede","mode":"cfb1","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[138,243,243,157,101,246,7,22]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[168,219,57,27,60,202,102,14,55,228,20,130,66,173,237,170]}},{"name":"2key3descfb","algorithm":"des-
ede","mode":"cfb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[30,81,130,204,99,135,126,40]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[38,159,250,38,73,190,6,6,61,205,69,35,138,183,208,17]}},{"name":"2key3descbc","algorithm":"des-
ede","mode":"cbc","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[41,247,217,151,49,3,25,25]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[157,224,31,140,119,75,215,51,112,60,249,156,10,28,78,131]}},{"name":"3key3desecb","algorithm":"des-
ede","mode":"ecb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[162,35,65,121,135,42,251,158,70,98,116,74,208,37,16,175]}},{"name":"3key3desofb","algorithm":"des-ede3","mode":"ofb","padding":"PKCS","keysize":"192","iv":
{"type":"Buffer","data":[212,61,100,85,110,33,71,166]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[238,63,249,159,161,205,114,29,112,4,86,239,169,151,191,140,244,4,208,176,80,94,255,240]}},{"name":"3key3descfb8","algorithm":"des-ede3","mode":"cfb8","padding":"PKCS","keysize":"192","iv":
{"type":"Buffer","data":[90,85,100,195,202,41,139,0]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[11,159,111,88,62,234,216,209,52,176,211,208,67,26,128,31,63,29,142,148,25,156,143,63]}},{"name":"3key3descfb1","algorithm":"des-ede3","mode":"cfb1","padding":"PKCS","keysize":"192","iv":
{"type":"Buffer","data":[82,87,175,128,242,57,182,181]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[192,221,238,89,204,225,44,199,142,67,16,248,170,64,197,201,204,97,2,248,17,215,55,123]}},{"name":"3key3descfb","algorithm":"des-ede3","mode":"cfb","padding":"PKCS","keysize":"192","iv":
{"type":"Buffer","data":[73,13,205,96,165,74,179,9]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[120,74,191,135,236,67,226,17,80,213,111,177,239,182,240,182,48,35,82,15,76,224,196,154]}},{"name":"3key3descbc","algorithm":"des-ede3","mode":"cbc","padding":"PKCS","keysize":"192","iv":
{"type":"Buffer","data":[41,214,25,9,106,119,209,101]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[159,242,184,120,251,176,122,74,51,116,36,97,213,97,171,88,50,143,139,36,60,102,87,32]}},{"name":"aes128ecb","algorithm":"aes","mode":"ecb","padding":"PKCS","keysize":"128","iv":
{"type":"Buffer","data":[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[139,188,185,105,48,21,240,126,41,254,91,176,248,38,147,222]}},{"name":"aes128ofb","algorithm":"aes","mode":"ofb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":
[158,115,88,205,38,159,231,143,197,157,78,254,71,171,143,28]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[71,65,108,205,129,140,58,241,127,78,80,127,0,249,8,162]}},{"name":"aes128cfb8","algorithm":"aes","mode":"cfb8","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":
[110,148,67,51,71,207,160,44,27,235,189,105,105,64,74,216]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[37,29,61,161,225,148,249,17,224,154,112,192,235,198,142,245]}},{"name":"aes128cfb1","algorithm":"aes","mode":"cfb1","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":
[31,111,4,148,250,199,163,84,5,95,184,169,97,212,224,210]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[200,141,249,194,212,199,48,138,80,58,80,226,235,104,197,149]}},{"name":"aes128cfb","algorithm":"aes","mode":"cfb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":
[90,251,88,200,253,217,79,28,242,91,38,158,55,186,197,129]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[206,37,191,59,91,24,234,60,248,105,133,26,110,31,192,39]}},{"name":"aes128cbc","algorithm":"aes","mode":"cbc","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":
[7,172,184,158,196,119,94,217,92,94,49,61,46,140,12,203]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[120,24,188,46,19,176,120,238,51,153,146,232,182,98,250,57]}},{"name":"aes128ctr","algorithm":"aes","mode":"ctr","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":
[248,217,153,68,48,238,23,37,0,63,208,153,164,204,100,71]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[243,65,181,79,112,99,72,43,62,226,128,95,227,24,164,216]}},{"name":"aes128cgcm","algorithm":"aes","mode":"gcm","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":
[79,62,139,238,131,3,116,55,124,141,131,28,126,1,119,183]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[131,12,224,231,245,150,61,223,128,149,156,184,212,223,104,145]}},{"name":"aes192ecb","algorithm":"aes","mode":"ecb","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":
[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[242,134,91,204,246,60,253,174,161,172,246,150,89,163,88,67,155,99,80,95,255,189,252,58]}},{"name":"aes192ofb","algorithm":"aes","mode":"ofb","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":
[19,232,105,63,12,207,110,155,33,16,164,53,63,34,36,78]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[218,216,79,200,185,74,117,87,25,200,230,162,5,193,85,247,196,166,70,110,125,180,9,41]}},{"name":"aes192cfb8","algorithm":"aes","mode":"cfb8","padding":"PKCS","keysize":"192","iv":
{"type":"Buffer","data":[80,174,86,53,123,5,206,223,213,233,155,160,57,8,229,38]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[158,114,152,27,56,71,54,77,85,136,120,38,84,22,193,163,85,96,203,233,204,201,2,238]}},
{"name":"aes192cfb1","algorithm":"aes","mode":"cfb1","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[67,10,19,35,2,7,203,145,44,194,96,150,47,55,207,143]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":
[127,254,65,128,16,47,200,99,118,88,253,179,221,111,45,207,117,41,28,144,111,125,204,110]}},{"name":"aes192cfb","algorithm":"aes","mode":"cfb","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[153,183,245,112,29,76,252,249,228,144,11,157,11,208,66,149]},"lifetime":"1","autorotate":"true","value":
{"type":"Buffer","data":[108,174,180,144,210,148,85,56,143,38,64,172,190,147,14,20,64,98,53,9,185,50,213,62]}},{"name":"aes192cbc","algorithm":"aes","mode":"cbc","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":
[187,191,208,111,62,111,97,2,127,117,81,108,73,233,165,164]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[157,182,97,227,75,32,8,24,108,238,223,163,13,245,170,127,46,54,176,7,59,60,133,197]}},{"name":"aes192ctr","algorithm":"aes","mode":"ctr","padding":"PKCS","keysize":"192","iv":
{"type":"Buffer","data":[181,17,106,194,246,168,24,211,149,143,56,52,126,160,194,219]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[188,185,38,182,42,232,103,177,89,230,151,179,250,132,15,65,159,111,74,250,222,190,10,188]}},
{"name":"aes192gcm","algorithm":"aes","mode":"gcm","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[142,154,252,183,252,176,244,26,208,189,113,142,17,239,86,153]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":
[188,45,170,82,230,172,142,35,13,252,194,253,40,189,47,121,47,239,71,34,203,154,89,103]}},{"name":"aes256ecb","algorithm":"aes","mode":"ecb","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":
[14,131,139,235,26,154,4,125,160,81,65,198,166,198,21,183,215,222,161,39,241,191,15,232,143,210,15,249,113,39,63,97]}},{"name":"aes256ofb","algorithm":"aes","mode":"ofb","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":
[88,227,42,42,81,242,247,202,161,198,166,174,49,221,228,251]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[220,32,131,182,36,246,168,240,73,139,1,195,162,109,22,149,9,196,46,212,3,118,185,79,131,94,15,106,119,26,204,9]}},
{"name":"aes256cfb8","algorithm":"aes","mode":"cfb8","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":[212,227,106,50,38,233,21,213,247,64,227,194,190,230,84,25]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":
[68,158,94,78,45,16,190,218,74,101,52,241,41,0,54,244,36,1,126,1,18,115,131,44,177,95,1,202,106,118,87,235]}},{"name":"aes256cfb1","algorithm":"aes","mode":"cfb1","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":
[128,29,28,58,218,249,25,39,80,102,240,218,63,219,144,108]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[63,139,170,235,127,136,87,0,160,169,53,55,251,150,162,83,36,184,227,106,192,84,183,170,245,151,42,31,208,75,196,19]}},
{"name":"aes256cfb","algorithm":"aes","mode":"cfb","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":[166,18,170,185,148,51,46,161,248,237,224,185,197,245,246,58]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":
[15,148,32,110,43,213,161,57,41,110,20,40,89,251,251,190,32,247,130,195,224,232,240,212,32,222,89,146,245,84,120,207]}},{"name":"aes256cbc","algorithm":"aes","mode":"cbc","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":
[91,61,246,75,80,72,253,127,241,75,12,107,137,37,24,129]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[156,194,233,239,209,237,161,168,22,14,43,250,157,66,67,198,228,215,22,106,100,16,109,182,55,97,111,203,89,184,203,249]}},
{"name":"aes256ctr","algorithm":"aes","mode":"ctr","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":[14,117,40,79,130,109,189,12,93,179,106,135,18,37,164,169]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":
[80,159,225,235,102,108,10,187,164,210,49,245,49,38,19,204,174,238,123,178,34,215,93,169,117,6,70,73,50,180,164,69]}},{"name":"aes256gcm","algorithm":"aes","mode":"gcm","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":
[181,96,7,113,231,227,30,132,117,114,27,179,198,21,59,89]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[107,244,201,185,5,196,241,95,64,136,202,17,158,4,212,222,242,204,92,105,210,132,67,96,103,51,95,68,122,143,222,188]}}]
```

Figure 13: Viewing information for all SSM application keys

Next, the master key rotation functionality is tested. The SSM requires a new 2-part master key, which it will validate against the conditions outlined in section 3.3.3. As with any admin operation, the current master key is required.

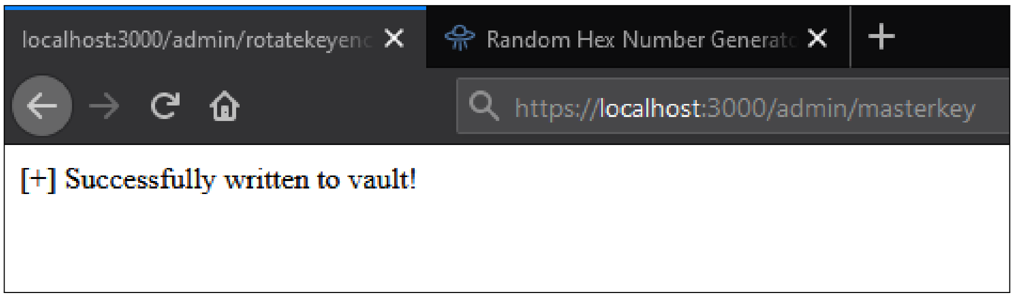

localhost:3000/admin/masterkey
Random Hex Number Generato
https://localhost:3000/admin/masterkey

Change Master Key

New Master key 1	3baa17bc6ec0be4aaf134
New Master key 2	b75d2ccb03389acc092ba
Master key 1	d4bc80ecf74a49aa45e12
Master key 2	6da7ca22a13cbfdee7bd6

SSM Admin Menu

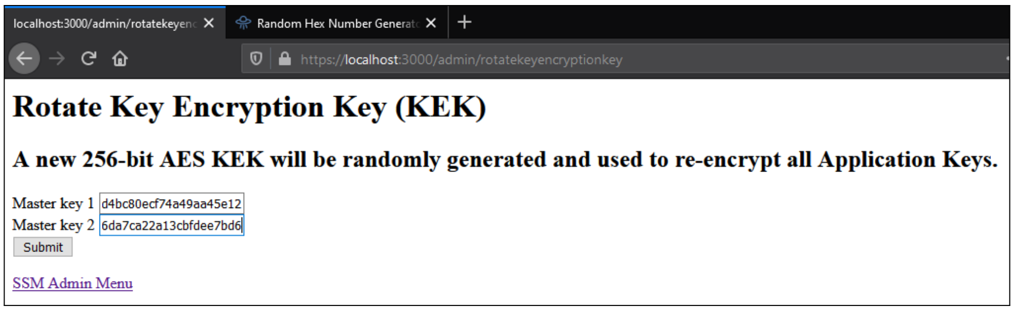
Figure 14: Change master key form


localhost:3000/admin/rotatekeyenc
Random Hex Number Generato
https://localhost:3000/admin/masterkey

[+] Successfully written to vault!

Figure 15: Master key rotation is confirmed

KEK rotation is very similar to master key rotation. However, the KEK is always generated using cryptographic pseudo-random functions within the SSM, and encrypted using AES-256-CBC. Therefore, only the master key is required to trigger a KEK rotation.


localhost:3000/admin/rotatekeyenc
Random Hex Number Generat
https://localhost:3000/admin/rotatekeyencryptionkey

Rotate Key Encryption Key (KEK)

A new 256-bit AES KEK will be randomly generated and used to re-encrypt all Application Keys.

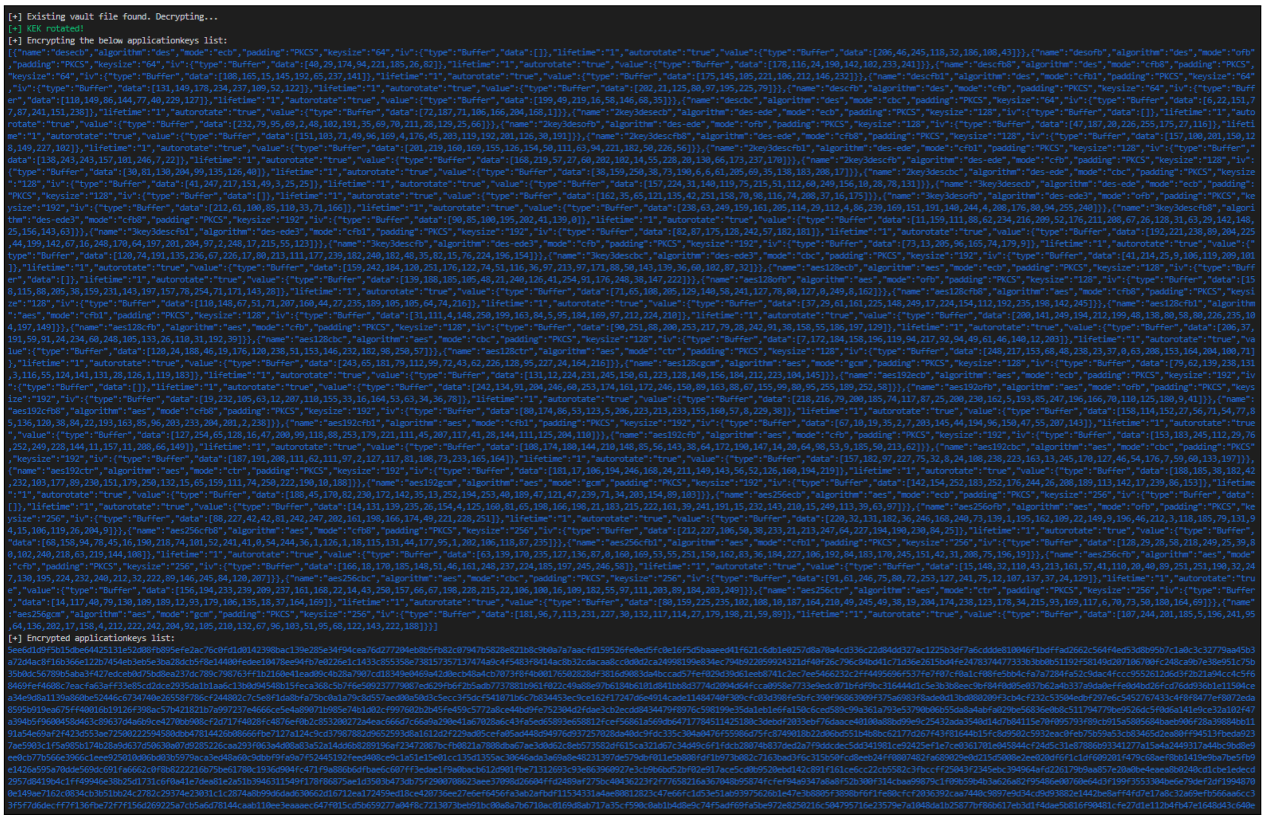
Master key 1 d4bc80ecf74a49aa45e12
Master key 2 6da7ca22a13cbfdee7bd6

Submit

SSM Admin Menu

Figure 16: Form for KEK rotation

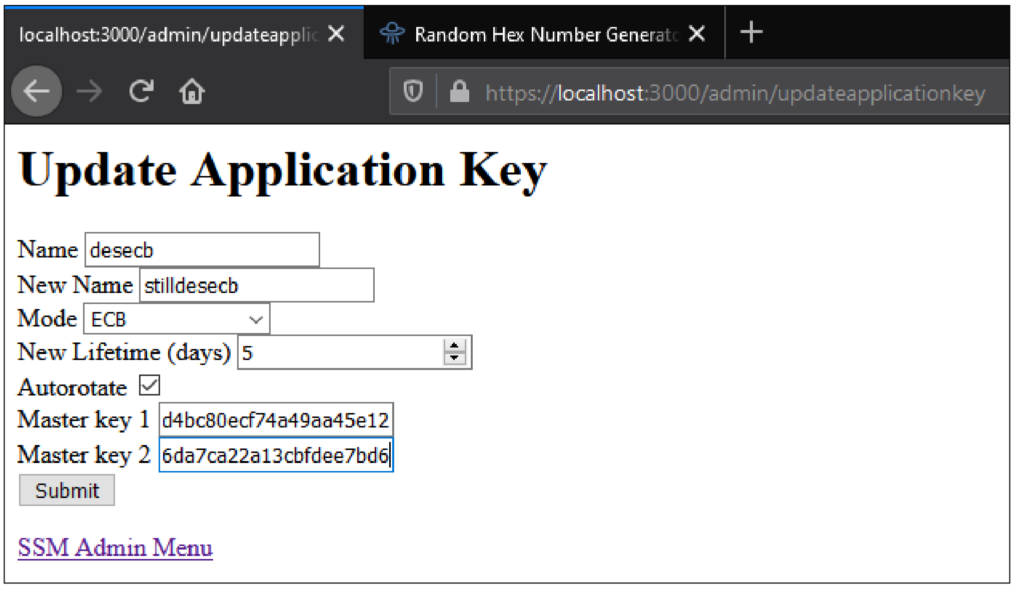
The SSM generates a new KEK and derives new KEK_1 and KEK_2 keys. All application keys are fetched from the vault, and re-encrypted and HMACed using the newly derived keys.



```
[+] Existing vault file found. Decrypting...
[+] KEK rotated!
[+] Encrypting the below applicationkeys list:
[{"name":"desecb","algorithm":"des","mode":"ecb","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data":[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[206,46,245,118,32,186,108,43]}},{"name":"desofb","algorithm":"des","mode":"ofb"
,"padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data":[40,29,174,94,221,185,26,82]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[178,116,24,190,142,102,233,241]}},{"name":"descfb8","algorithm":"des","mode":"cfb8","padding":"PKCS",
"keysize":"64","iv":{"type":"Buffer","data":[108,165,15,145,192,65,237,141]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[175,145,105,221,106,212,146,232]}},{"name":"descfb1","algorithm":"des","mode":"cfb1","padding":"PKCS","keysize":"64"
,"iv":{"type":"Buffer","data":[131,149,178,234,237,109,52,122]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[202,21,125,80,97,195,225,79]}},{"name":"descfb","algorithm":"des","mode":"cfb","padding":"PKCS","keysize":"64","iv":{"type":"Buff
er","data":[110,149,86,144,77,40,229,127]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[199,49,219,16,58,146,68,35]}},{"name":"descbc","algorithm":"des","mode":"cbc","padding":"PKCS","keysize":"64","iv":{"type":"Buffer","data":[6,22,151,7
7,87,241,151,238]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[72,187,71,106,166,204,168,1]}},{"name":"2key3desecb","algorithm":"des-ede","mode":"ecb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[]},"lifetime":"1","auto
rotate":"true","value":{"type":"Buffer","data":[232,79,95,69,2,48,102,191,35,69,70,211,28,129,25,66]}},{"name":"2key3desofb","algorithm":"des-ede","mode":"ofb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[47,187,20,226,255,175,27,116]},"lifeti
me":"1","autorotate":"true","value":{"type":"Buffer","data":[151,103,71,49,96,169,4,176,45,203,119,192,201,126,30,191]}},{"name":"2key3descfb8","algorithm":"des-ede","mode":"cfb8","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[157,100,201,150,12
8,149,227,102]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[201,219,160,169,155,126,154,50,111,63,94,221,182,50,226,56]}},{"name":"2key3descfb1","algorithm":"des-ede","mode":"cfb1","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","
data":[138,243,243,157,101,246,7,22]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[168,219,57,27,60,202,102,14,55,228,20,130,66,173,237,170]}},{"name":"2key3descfb","algorithm":"des-ede","mode":"cfb","padding":"PKCS","keysize":"128","iv":
{"type":"Buffer","data":[30,81,130,204,99,135,126,40]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[38,159,250,38,73,190,6,6,61,205,69,35,138,183,208,17]}},{"name":"2key3descbc","algorithm":"des-ede","mode":"cbc","padding":"PKCS","keysize
":"128","iv":{"type":"Buffer","data":[41,247,217,151,49,3,25,25]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[157,224,31,140,119,75,215,51,112,60,249,156,10,28,78,131]}},{"name":"3key3desecb","algorithm":"des-ede","mode":"ecb","padding":
"PKCS","keysize":"128","iv":{"type":"Buffer","data":[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[162,35,65,121,135,42,251,158,70,98,116,74,208,37,16,175]}},{"name":"3key3desofb","algorithm":"des-ede3","mode":"ofb","padding":"PKCS","ke
ysize":"192","iv":{"type":"Buffer","data":[212,61,100,85,110,33,71,166]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[238,63,249,159,161,205,114,29,112,4,86,239,169,151,191,140,244,4,208,176,80,94,255,240]}},{"name":"3key3descfb8","algori
thm":"des-ede3","mode":"cfb8","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[90,85,100,195,202,41,139,0]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[11,159,111,88,62,234,216,209,52,176,211,208,67,26,128,31,63,29,142,148,
25,156,143,63]}},{"name":"3key3descfb1","algorithm":"des-ede3","mode":"cfb1","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[82,87,175,128,242,57,182,181]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[192,221,238,89,204,225
,44,199,142,67,16,248,170,64,197,201,204,97,2,248,17,215,55,123]}},{"name":"3key3descfb","algorithm":"des-ede3","mode":"cfb","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[73,13,205,96,165,74,179,9]},"lifetime":"1","autorotate":"true","value":{"
type":"Buffer","data":[120,74,191,135,236,67,226,17,80,213,111,177,239,182,240,182,48,35,82,15,76,224,196,154]}},{"name":"3key3descbc","algorithm":"des-ede3","mode":"cbc","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[41,214,25,9,106,119,209,101
]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[159,242,184,120,251,176,122,74,51,116,36,97,213,97,171,88,50,143,139,36,60,102,87,32]}},{"name":"aes128ecb","algorithm":"aes","mode":"ecb","padding":"PKCS","keysize":"128","iv":{"type":"Buff
er","data":[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[139,188,185,105,48,21,240,126,41,254,91,176,248,38,147,222]}},{"name":"aes128ofb","algorithm":"aes","mode":"ofb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[15
8,115,88,205,38,159,231,143,197,157,78,254,71,171,143,28]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[71,65,108,205,129,140,58,241,127,78,80,127,0,249,8,162]}},{"name":"aes128cfb8","algorithm":"aes","mode":"cfb8","padding":"PKCS","keysi
ze":"128","iv":{"type":"Buffer","data":[110,148,67,51,71,207,160,44,27,235,189,105,105,64,74,216]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[37,29,61,161,225,148,249,17,224,154,112,192,235,198,142,245]}},{"name":"aes128cfb1","algorithm
":"aes","mode":"cfb1","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[31,111,4,148,250,199,163,84,5,95,184,169,97,212,224,210]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[200,141,249,194,212,199,48,138,80,58,80,226,235,10
4,197,149]}},{"name":"aes128cfb","algorithm":"aes","mode":"cfb","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[90,251,88,200,253,217,79,28,242,91,38,158,55,186,197,129]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[206,37,
191,59,91,24,234,60,248,105,133,26,110,31,192,39]}},{"name":"aes128cbc","algorithm":"aes","mode":"cbc","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[7,172,184,158,196,119,94,217,92,94,49,61,46,140,12,203]},"lifetime":"1","autorotate":"true","va
lue":{"type":"Buffer","data":[120,24,188,46,19,176,120,238,51,153,146,232,182,98,250,57]}},{"name":"aes128ctr","algorithm":"aes","mode":"ctr","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[248,217,153,68,48,238,23,37,0,63,208,153,164,204,100,71]
},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[243,65,181,79,112,99,72,43,62,226,128,95,227,24,164,216]}},{"name":"aes128cgcm","algorithm":"aes","mode":"gcm","padding":"PKCS","keysize":"128","iv":{"type":"Buffer","data":[79,62,139,238,131
,3,116,55,124,141,131,28,126,1,119,183]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[131,12,224,231,245,150,61,223,128,149,156,184,212,223,104,145]}},{"name":"aes192ecb","algorithm":"aes","mode":"ecb","padding":"PKCS","keysize":"192","iv
":{"type":"Buffer","data":[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[242,134,91,204,246,60,253,174,161,172,246,150,89,163,88,67,155,99,80,95,255,189,252,58]}},{"name":"aes192ofb","algorithm":"aes","mode":"ofb","padding":"PKCS","keys
ize":"192","iv":{"type":"Buffer","data":[19,232,105,63,12,207,110,155,33,16,164,53,63,34,36,78]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[218,216,79,200,185,74,117,87,25,200,230,162,5,193,85,247,196,166,70,110,125,180,9,41]}},{"name":
"aes192cfb8","algorithm":"aes","mode":"cfb8","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[80,174,86,53,123,5,206,223,213,233,155,160,57,8,229,38]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[158,114,152,27,56,71,54,77,8
5,136,120,38,84,22,193,163,85,96,203,233,204,201,2,238]}},{"name":"aes192cfb1","algorithm":"aes","mode":"cfb1","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[67,10,19,35,2,7,203,145,44,194,96,150,47,55,207,143]},"lifetime":"1","autorotate":"true
","value":{"type":"Buffer","data":[127,254,65,128,16,47,200,99,118,88,253,179,221,111,45,207,117,41,28,144,111,125,204,110]}},{"name":"aes192cfb","algorithm":"aes","mode":"cfb","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[153,183,245,112,29,76
,252,249,228,144,11,157,11,208,66,149]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[108,174,180,144,210,148,85,56,143,38,64,172,190,147,14,20,64,98,53,9,185,50,213,62]}},{"name":"aes192cbc","algorithm":"aes","mode":"cbc","padding":"PKCS"
,"keysize":"192","iv":{"type":"Buffer","data":[187,191,208,111,62,111,97,2,127,117,81,108,73,233,165,164]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[157,182,97,227,75,32,8,24,108,238,223,163,13,245,170,127,46,54,176,7,59,60,133,197]}},
{"name":"aes192ctr","algorithm":"aes","mode":"ctr","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[181,17,106,194,246,168,24,211,149,143,56,52,126,160,194,219]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[188,185,38,182,42
,232,103,177,89,230,151,179,250,132,15,65,159,111,74,250,222,190,10,188]}},{"name":"aes192gcm","algorithm":"aes","mode":"gcm","padding":"PKCS","keysize":"192","iv":{"type":"Buffer","data":[142,154,252,183,252,176,244,26,208,189,113,142,17,239,86,153]},"lifetime
":"1","autorotate":"true","value":{"type":"Buffer","data":[188,45,170,82,230,172,142,35,13,252,194,253,40,189,47,121,47,239,71,34,203,154,89,103]}},{"name":"aes256ecb","algorithm":"aes","mode":"ecb","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":
[]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[14,131,139,235,26,154,4,125,160,81,65,198,166,198,21,183,215,222,161,39,241,191,15,232,143,210,15,249,113,39,63,97]}},{"name":"aes256ofb","algorithm":"aes","mode":"ofb","padding":"PKCS","ke
ysize":"256","iv":{"type":"Buffer","data":[88,227,42,42,81,242,247,202,161,198,166,174,49,221,228,251]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[220,32,131,182,36,246,168,240,73,139,1,195,162,109,22,149,9,196,46,212,3,118,185,79,131,9
4,15,106,119,26,204,9]}},{"name":"aes256cfb8","algorithm":"aes","mode":"cfb8","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":[212,227,106,50,38,233,21,213,247,64,227,194,190,230,84,25]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer",
"data":[68,158,94,78,45,16,190,218,74,101,52,241,41,0,54,244,36,1,126,1,18,115,131,44,177,95,1,202,106,118,87,235]}},{"name":"aes256cfb1","algorithm":"aes","mode":"cfb1","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":[128,29,28,58,218,249,25,39,8
0,102,240,218,63,219,144,108]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[63,139,170,235,127,136,87,0,160,169,53,55,251,150,162,83,36,184,227,106,192,84,183,170,245,151,42,31,208,75,196,19]}},{"name":"aes256cfb","algorithm":"aes","mode"
:"cfb","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":[166,18,170,185,148,51,46,161,248,237,224,185,197,245,246,58]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[15,148,32,110,43,213,161,57,41,110,20,40,89,251,251,190,32,24
7,130,195,224,232,240,212,32,222,89,146,245,84,120,207]}},{"name":"aes256cbc","algorithm":"aes","mode":"cbc","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":[91,61,246,75,80,72,253,127,241,75,12,107,137,37,24,129]},"lifetime":"1","autorotate":"tru
e","value":{"type":"Buffer","data":[156,194,233,239,209,237,161,168,22,14,43,250,157,66,67,198,228,215,22,106,100,16,109,182,55,97,111,203,89,184,203,249]}},{"name":"aes256ctr","algorithm":"aes","mode":"ctr","padding":"PKCS","keysize":"256","iv":{"type":"Buffer
","data":[14,117,40,79,130,109,189,12,93,179,106,135,18,37,164,169]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[80,159,225,235,102,108,10,187,164,210,49,245,49,38,19,204,174,238,123,178,34,215,93,169,117,6,70,73,50,180,164,69]}},{"name"
:"aes256gcm","algorithm":"aes","mode":"gcm","padding":"PKCS","keysize":"256","iv":{"type":"Buffer","data":[181,96,7,113,231,227,30,132,117,114,27,179,198,21,59,89]},"lifetime":"1","autorotate":"true","value":{"type":"Buffer","data":[107,244,201,185,5,196,241,95
,64,136,202,17,158,4,212,222,242,204,92,105,210,132,67,96,103,51,95,68,122,143,222,188]}}]
[+] Encrypted applicationkeys list:
5ee6d1d9f5b15dbe64425131e52d08fb895efe2ac76c0fd1d0142398bac139e285e34f94cea76d277204eb8b5fb82c07947b5828e821b8c9b0a7a7aacfd159526fe0ed5fc0e16f5d5baaeed41f621c6db1e0257d8a70a4cd336c22d84dd327ac1225b3df7a6cddde810046f1bdffad2662c564f4ed53d8b95b7c1a0c3c32779aa45b3
a72d4ac8f16b366e122b7454eb3eb5e3ba28dcb5f8e14400fedee10478ee94fb7e0226e1c1433c855358e738157357137474a9c4f5483f8414ac8b32cdacaa8cc0d0d2ca24998199e834ec794b922059924321df40f26c796c84bd41c71d36e2615bd4fe2478374477333b3bb0b51192f58149d207106700fc248ca9b7e38e951c75b
35b0dc56789b5aba3f427edceb0d75bd8ea237dc789c798763ff1b2160e41ead09c4b28a7907cd18349e0469a42d0ecb48a4cb7073f8f4b00176502828df3816d9083da4bccad57fef029d39d61eeb8741c2ec7ee5466232c2ff4495696f537fe7f07cf0a1cf08fe5bb4cfa7a7284fa52c9dac4fccc9552612d6d3f2b21a94cc4c5f6
8469fef4608c7eacfa63aff33e85cd2dce2935da1b1aa6c13b0d94548b15feca368c5b7f6e509237779087ed629fb6f2b5adb7737881b961f022c49a88e97b6184b6101d841bb8d3774d2094d64fcca0958e7733e9edc071bfdf9bc316444d1c5e3b3b8eec9bf84f0d05e037b62a4b337a9da0effe0d4bd26fcd76dd936b1e11504ce
a34e9d8a1139a860be52446c6734740e26558f786cf244802c7c5e8f1da8bfa75bc0a1a79c8d557aed00a50d3c5ecc3f5dcf541071b6c7b834453ec9ce162f17247d6e4914cade11484740f309cfc03d398fe5bfc390f96863909f375a6983f8ade0d13bd080209f3cb4cf232c53504edbf297e6c5452767433c4f8f8477ef8072eda
8595b919ea675ff40016b19126f398ac57b421821b7a997237e4666ce5e4a89071b985e74b1d02cf997602b2b45fe459c5772a8ce44bd9fe752304d2fdae3cb2ecdd8434479f8976c598199e35da1eb1e6fa150c6ced589c99a361a793e53790b06b55da8a4abfa029be56836e0b8c511794779be9526dc5f0d6a141e9ce32a102f47
a394b5f9600458d463c89637d4a6b9ce4270bb908cf2d717f4028fc4876ef0b2c853200272a4eac666d7c66a9a290e41a67028a6c43fa5ed65893e658812fcef56861a569db64717784511425180c3debdf2033ebf76daace40100a88bd99e9c25432ada3540d14d7b84115e70f095793f89cb915a5805684baeb906f28a39884bb11
91a54e69af2f423d553ae72500222594580dbb47814426b08666fbe7127a124c9cd37987882d9652593d8a1612d2f229ad05cefa05ad448d94976d937257028da40dc9fdc335c304a0476f55986d75fc8749018b22d06bd551b4b8bc62177d267f43f81644b15fc8d9502c5932eac0feb75b59a53cb83465d2ea80ff94513fbeda923
7ae5903c1f5a985b174b28a9d637d50630a07d9285226caa293f063a4d08a83a52a14dd6b8289196af23472087bcfb0821a7808dba67ae3d0d62c8eb573582df615ca321d67c34d49c6f1fdcb28074b837ded2a7f9ddcdec5dd341981ce92425ef1e7ce0361701e045844cf24d5c31e87886b93341277a15a4a2449317a44bc9bd8e9
ee0cb77b566e3966c1eee925010d06bd03b5979aca3ed48a60c9dbbf9fa9a7f52445192feed408ce9c1a51e15e01cc135d1355ac30646ada3a69a8e48231397de579dbf011e5b808fdf1b973b082c7163bad3f6c315b50fcd8eeb24ff0807482fa689029e0d215d5008e2ee020df6f1c1df609201f479c68aef8bb1419e9ba7be5fb9
e1426a595a70dde569dc691fa6662c0f8b8222216b75be61780c1936d904fc471f9a886b6dfbae6c607ff3edae1f9a0bacb612d901fbe71312693c93e863960927e3cb9b6bd52bf02e917ace5cd0b9520ebd142c891f161ce6cc22cb5582c3fbccff25043f2345ebc394964afd226179b9aa857e20a0be4eaea8b0240cd1cbe1edecd
2957d8419b4c1ff49946e38b25d1731c6f0a41e7dea81e2a51b3946311549f178f86875ae1d3503b473db75f2900708623aee37098d26604ffd2489af275bc40436223f2f77658216a367048b95874fcfef94a9347a8a8f52b300f314cbaa99879c1f09b59b4b3a626a82f95486e00760e64d3f199f3553304be6e79def2df1994870
0e149ae7162c0834cb3b51bb24c2782c29374e23031c1c2874a8b99d6dad630662d16712ea172459ed18ce420736ee27e6ef6456fa3ab2afbdf11534331a4ae80812823c47e66fc1d53e51ab93975626b1e47e3b8805f3898bf6f1fe80cfcf2036392caa7440c9897e9d34cd9d93882e1442be8aff4fd7e17a8c32a69efb566aa6cc3
3f5f7d6decff7f136fbe72f7f156d269225a7cb5a6d78144caab110ee3eaaaec647f015cd5b659277a04f8c7213073beb91bc00a8a7b6710ac0169d8ab717a35cf590c0ab1b4d8e9c74f5adf69fa5be972e8250216c504795716e23579e7a1048da1b25877bf86b617eb3d1f4dae5b816f90481cfe27d1e112b4fb47e1648d43c640e
```

Figure 17: SSM KEK rotation under the hood

Next, updating an application key parameters is demonstrated. I rename the DES-ECB application key and change its lifetime from 1 day to 5 days.


localhost:3000/admin/updateapplic
Random Hex Number Generatc
https://localhost:3000/admin/updateapplicationkey

Update Application Key

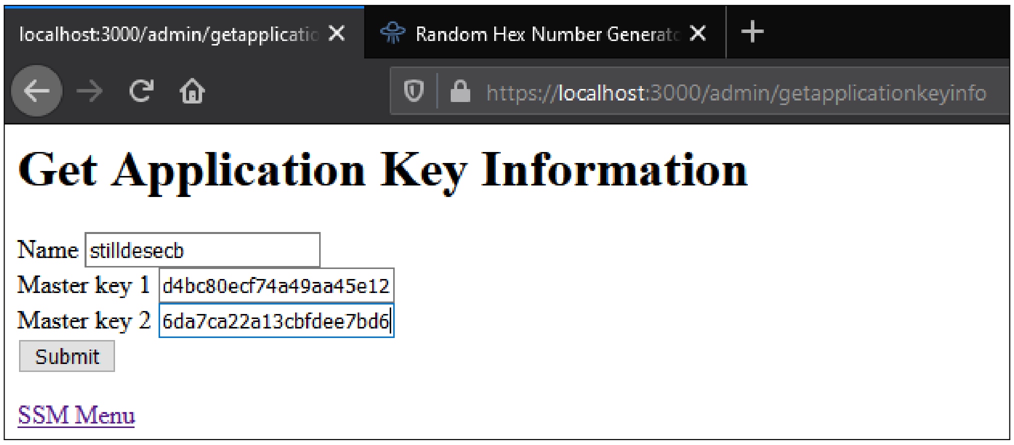
Name	desecb
New Name	stilldesecb
Mode	ECB
New Lifetime (days)	5
Autorotate	✓
Master key 1	d4bc80ecf74a49aa45e12
Master key 2	6da7ca22a13cbfdee7bd6

Submit

SSM Admin Menu

Figure 18: Form for updating an application key

To check whether the application key has successfully been updated, I get the application key's information using its new name.


localhost:3000/admin/getapplicatic
Random Hex Number Generatc
https://localhost:3000/admin/getapplicationkeyinfo

Get Application Key Information

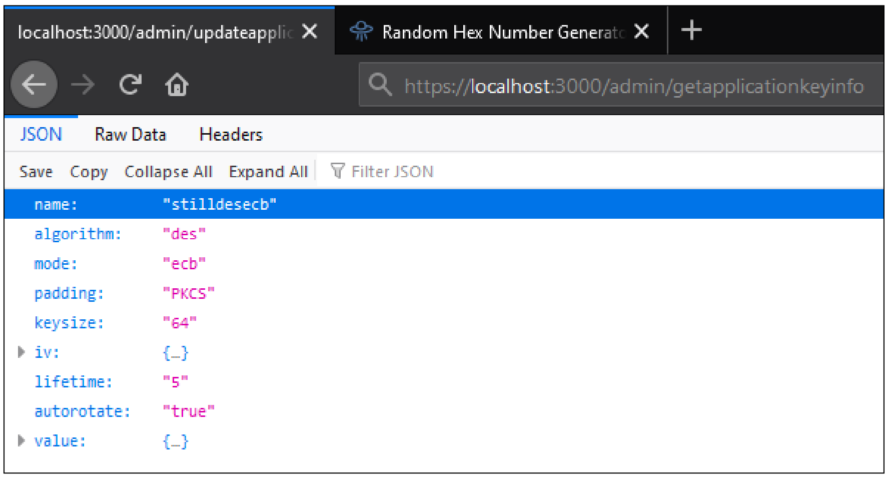
Name	stilldesecb
Master key 1	d4bc80ecf74a49aa45e12
Master key 2	6da7ca22a13cbfdee7bd6

Submit

SSM Menu

Figure 19: Searching a newly updated application key by name

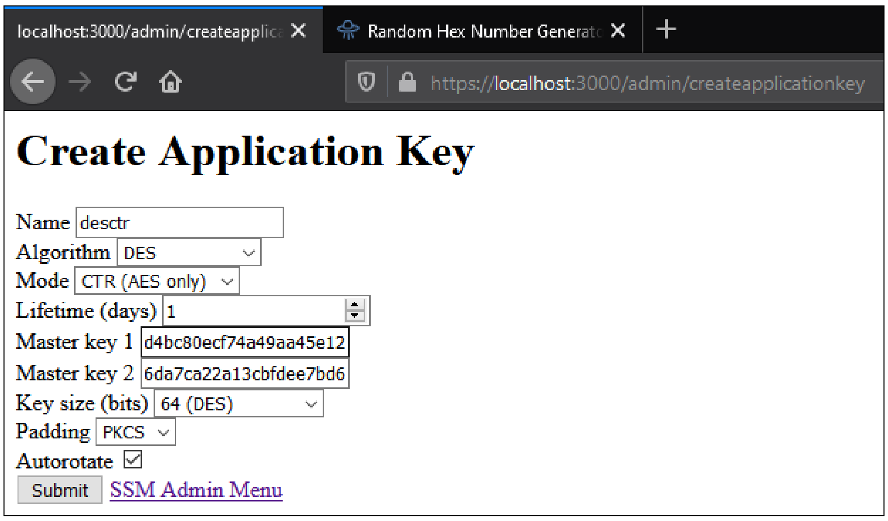
The JSON application key object returned shows that both the name and lifetime have been successfully updated.


localhost:3000/admin/updateapplic
Random Hex Number Generato
https://localhost:3000/admin/getapplicationkeyinfo
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
name:        "stilldesecb"
algorithm:   "des"
mode:        "ecb"
padding:     "PKCS"
keysize:     "64"
iv:          {…}
lifetime:    "5"
autorotate:  "true"
value:       {…}
```

Figure 20: Confirming an application key can be correctly updated

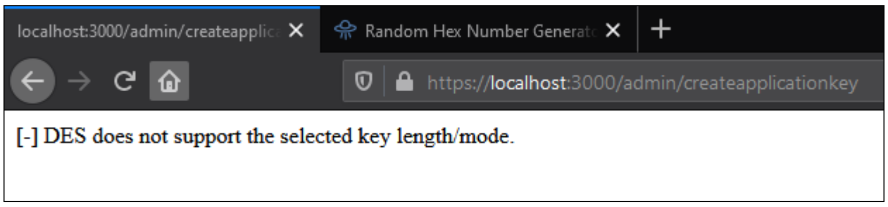
Finally, I demonstrate that the SSM stops an admin from creating an application key with incompatible parameters. For example, the Crypto API does not support DES in counter mode. Therefore, the SSM should reject the admin command and not apply any changes to the vault.


localhost:3000/admin/createapplic
Random Hex Number Generato
https://localhost:3000/admin/createapplicationkey
Submit
SSM Admin Menu

Create Application Key

Name	desctr
Algorithm	DES
Mode	CTR (AES only)
Lifetime (days)	1
Master key 1	d4bc80ecf74a49aa45e12
Master key 2	6da7ca22a13cbfdee7bd6
Key size (bits)	64 (DES)
Padding	PKCS
Autorotate	☑

Figure 21: Attempting to create an application key using an illegal configuration.

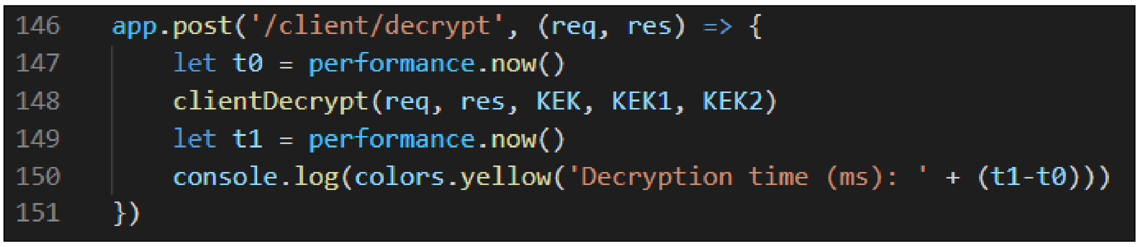

localhost:3000/admin/createapplic
Random Hex Number Generato
https://localhost:3000/admin/createapplicationkey

[-] DES does not support the selected key length/mode.

Figure 22: Key creation is unsuccessful, and error is handled appropriately.

6 Comparing Block Cipher Performance

The performance of DES, 3DES, and AES was measured accurately by finding the difference between the time, before and after encryption/decryption in milliseconds. A long string containing the first 3000 words from Quentin Tarantino's Pulp Fiction was used as data to encrypt/decrypt.



```
app.post('/client/decrypt', (req, res) => {
    let t0 = performance.now()
    clientDecrypt(req, res, KEK, KEK1, KEK2)
    let t1 = performance.now()
    console.log(colors.yellow('Decryption time (ms): ' + (t1-t0)))
})
```

Figure 23: Accurately measuring decryption time

6.1 Tables of Results

6.1.1 Encryption

	ECB	OFB	CFB-8	CFB-1	CFB	CBC	CTR	GCM
DES	2.081	2.297	5.672	37.240	2.013	2.272	n/a	n/a
3DES	2.657	2.673	12.959	92.802	3.321	3.795	n/a	n/a
AES-128	2.127	1.595	2.321	9.274	2.052	2.241	1.359	2.086
AES-192	1.810	1.633	2.154	10.183	1.313	1.259	1.381	1.547
AES-256	1.622	1.679	2.958	10.999	2.061	1.650	1.769	1.655

Table 1: Application key encryption times in milliseconds

6.1.2 Decryption

	ECB	OFB	CFB-8	CFB-1	CFB	CBC	CTR	GCM
DES	2.319	2.845	5.779	39.516	2.224	2.272	n/a	n/a
3DES	3.012	3.347	12.568	96.587	3.428	4.758	n/a	n/a
AES-128	1.888	2.360	2.564	10.127	1.826	2.874	2.472	2.104
AES-192	1.825	2.020	2.801	10.521	2.244	2.014	2.876	2.891
AES-256	2.177	1.514	2.603	11.689	1.856	1.682	1.609	2.380

Table 2: Applicatiom key decryption times in milliseconds

6.2 Visualizing the Results

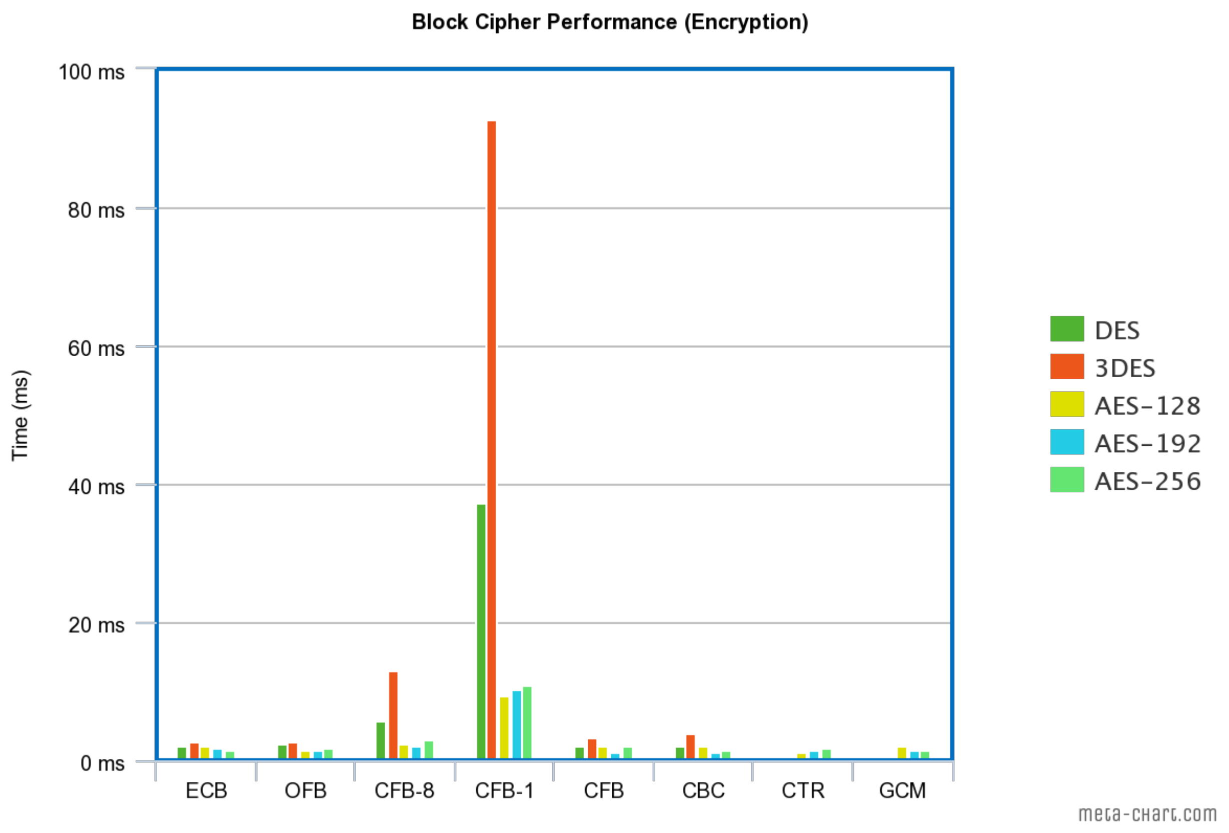


Figure 24: Bar plots comparing the different modes of operation across algorithms. CFB-1 is clearly the slowest mode of operation, and 3DES the slowest block cipher.

7 Enhancing the SSM Security

- A major security concern is the storing of the KEK within the SSM for as long as it is running. While this is arguably not as bad as storing the entire *applicationkeys* array, if a hacker gains access the the server, they are able to read the KEK. Knowing the KEK, it is trivial to fetch and decrypt the application keys from the vault. Timeouts can be implemented, which encrypt the KEK after a period of inactivity.
- Audit trails would greatly enhance the overall SSM security. These allow administrators to view a log of which keys were used at particular times, by which service/entity.
- The JSON vault file is made up of *key: value* pairs, where *key* directly describes *value*. This introduces a lack of obscurity, as it is clear that e.g., *KeyEncryptionKeyEncMK1* denotes the KEK, encrypted with MK_1. Therefore, if an attacker were to gain server-side access, they would immediately know what they are looking at. Ideally, the entire vault should be encrypted and HMACed. However, this would mean that if the vault is tampered with, then the HMAC would fail and render the entire vault unusable. A solution to this would be to implementH MACs on several levels, down to the *applicationkey.value* level.

- The NodeJS Crypto API supports a plethora of ciphers and hashing algorithms. Many of these (excluding any inherently weak algorithms e.g., MD5) should be included for a more comprehensive software security module. Digital signatures such as RSA would also be extremely beneficial for signing and verifying data.
- Many HSMs are able to provide resistance against physical tampering. This means that when an attacker attempts to take apart the hardware, sensitive data is wiped before it can be read. Similar behaviour can be implemented for the SSM, such that if the vault is moved or attempted to be read by any entity other than the SSM, data is wiped.
- Finally, the JSON vault is not resistant to simple deletion! It is possible for deliberate (or acceidental) deletion to occur, with no preventative mechanism in place. This should be addressed in order to improve the SSM security.

8 Key Management Policy for the Master Key

As outlined in section 3.2, the 2-part master is meant to be held by two different administrators who only know their part of the master key. This prevents Eve from gaining access to the SSM should she manage to acquire one of the two parts, and also ensures that a rogue admin is unable to make unapproved changes to the SSM. Key management is arguably the most crucial part of any cryptographic system. I outline a key management policy for the master key below:

- When created the master key must be random and validated properly (see 3.3.3).
- Each administrator should not know the value of the second key part.
- Master key parts must be stored securely, e.g., using a password manager like *LastPass*[4] and definitely not written on paper.
- The master key should be rotated every week.
- The keys should have a hard expiry date upon which rotation should be forced.
- Expired keys that have reach the end of their lifecycle should be destroyed.

[4]https://www.lastpass.com/